VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizace sociálních sítí
Visualization of Social Network

2014 Jan Kuba

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Kuba**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vizualizace sociálních sítí
Visualization of Social Networks

Zásady pro vypracování:

Analýza a vizualizace reálných dat reprezentujících rozsáhlé sítě různého typu (např. sociální sítě) je v současnosti v centru pozornosti širokého vědeckého společenství. Navzdory velkému počtu algoritmů pro kreslení (layout) sítí (grafů), existuje potřeba vhodně měnit zobrazení sítí, vizualizovat výsledky analýzy sítí, reprezentovat různé role síťových entit a vztahy mezi nimi atd. Hledání komunit v sítích a jejich vizualizace je jednou z dílčích oblastí, kterými se lze zabývat. Cílem práce je realizace podpory vizualizace vybraných vlastností komunit nalezených např. v sociální síti typu síť spolupráce (collaborative network) pomocí WebGL.

1. Seznamte se s jednotlivými typy algoritmů pro layout sítí i s aplikacemi pro jejich použití.
2. Seznamte se technologií WebGL.
3. Analyzujte požadavky na vizualizaci komunit a jejich vlastností.
4. Navrhněte a naimplementujte aplikaci pro vizualizaci vybraných vlastností komunit.

Seznam doporučené odborné literatury:

[1] Lima M.: Visual Complexity: Mapping Patterns of Information. Princeton Architectural Press, 2011, ISBN-10: 1568989369
[2] WebGL, http://www.khronos.org/registry/webgl/specs/1.0/
[3] Dle doporučení vedoucího bakalářské práce

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **RNDr. Eliška Ochodková, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě dne 7. května 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. května 2014 .....

Rád bych na tomto místě poděkoval paní RNDr. Elišce Ochodkové, Ph.D. a panu Mgr. Miloši Kudělkovi, Ph.D. za jejich ochotu, pomoc a odborné vedení při řešení bakalářské práce.

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Poděkování

Tato práce byla vypracována s podporou projektu Rozvoj lidských zdrojů ve výzkumu a vývoji moderních soft computingových metod a jejich praktického využití, reg. č. CZ.1.07/2.3.00/20.0072 podpořeného Operačním programem Vzdělávání pro konkurenceschopnost, financovaného ze strukturálních fondů EU a státního rozpočtu ČR.

Abstrakt

Hlavním cílem této práce je naimplementovat aplikaci, která by sloužila pro vizualizaci sítí ve 3D. Aplikace je implementována pro webovou platformu a to pomocí WebGL. Aplikace získává data pomocí webové služby, na kterou se aplikace umí napojit. Aplikace je schopna vizualizovat vybrané vlastnosti sítí, především komunity. Také je schopna zobrazit informace o jednotlivých vrcholech a komunitách sítě. Dalším cílem je se seznámit s aplikacemi, které jsou zaměřené na vizualizaci sítí a s algoritmy pro layout, které tyto aplikace pro vizualizaci používají.

Klíčová slova: WebGL, graf, síť, komunita, vizualizace, layout

Abstract

The main objective of this work is to implement an application which would be able to visualize networks in 3D. The application is implemented for a web platform and using WebGL. The application receives data via web service to which the application is able to connect. The application is able to visualize specific properties of networks, in particular communities. It is also able to view information about individual vertices and community of networks. Another aim is to get acquainted with the programs that are aimed at visualizing networks and algorithms for layout which these applications are used for visualization.

Keywords: WebGL, graph, network, community, visualization, layout

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
WebGL	–	Web Graphics Library
AJAX	–	Asynchronous Javascript and XML
XML	–	Extensible Markup Language
JSON	–	JavaScript Object Notation

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Sítěmi je možné modelovat spoustu reálných systémů. Analýza sítí je proto v dnešní době v centru pozornosti vědeckého společenství. Nezbytnou součástí analýzy sítí je její vizualizace. Vizualizace sítí nám pomáhá s analýzou a čtením informací z těchto sítí. Vizualizace sítě nám pomáhá pochopit jejich strukturu. Práce je zaměřená právě na onu vizualizaci a její součástí je naimplementovat aplikaci, která bude vizualizovat síť a její vlastnosti a informace o určitých objektech.

Kapitola Sítě je zaměřena na obecné pojmy problematiky sítí. Je zde vysvětleno, co to vlastně síť je, jaké mohou být druhy sítí a proč bychom se sítěmi měli zabývat. Kapitola se dále věnuje pojmu "komunita", pojem je zde vysvětlen a nastíněn způsob jejich hledání v sítích. Je zde i podkapitola, která se věnuje vybraným algoritmům pro layout. Jsou zde popsány jejich výhody a nevýhody. Další podkapitola se věnuje Sammonově projekci, což je metoda pro redukci dimenze. Tuto metodu využívá webová služba, která aplikaci poskytuje data o rozmístění vrcholů. V poslední části jsou ukázky vizualizace jedné datové kolekce různými algoritmy.

Kapitola WebGL je zaměřena na technologii, s pomocí které je naimplementována výsledná aplikace pro vizualizaci sítě. Je zde zmínka o jejím vývoji a její silných stránkách. Také jsou zde popsány kroky, které je nutné provézt pro vykreslení určitého objektu. Tyto kroky jsou důkladněji popsány v dalších částech kapitoly.

Kapitola Analýza a návrh aplikace je zaměřena na analýzu aplikace z pohledu softwarového inženýra. Je zde soupis funkčních požadavků na aplikaci a soupis použitých technologií a knihoven pro tvorbu aplikace. Můžeme zde také najít diagramy tříd s jednoduchým popisem každé třídy. Také je zde možno najít dva sekvenční diagramy a jejich popis.

Kapitola Implementace se zaměřuje na vybrané a zajímavé části implementace. Nejdříve se kapitola zaměří na způsob výběru objektů. Dalším tématem kapitoly je vizualizace. V této podkapitole bude zmíněno, které objekty pro vizualizaci byly použity a jak jsou tyto objekty vytvářeny. Také bude řečeno, jaký stínovací model byl použit pro stínování objektů. Další podkapitola se věnuje zobrazení informací. Je zde popsána komponenta, která pro to slouží. Dále bude popsáno, jakým způsobem jsou v aplikaci zobrazovány popisky vrcholů. Také bude popsáno, jakým způsobem je naimplementován grafický prvek, jenž dodává u vrcholů dojem jejich záře. Dále bude popsána animace grafu při jeho načtení a poslední část se bude věnovat propojení aplikace s webovou službou.

2 Sítě

Sítěmi nazýváme kolekce objektů zvané vrcholy, které jsou mezi sebou propojené linky neboli hranami. Takový systém vrcholů a hran pak dostává formu sítě. Místo pojmu síť je možno použít i název „graf“. Jako příklady sítí si můžeme uvést například internet – představit si jej jako síť navzájem propojených počítačů, sociální sítě – síť navzájem propojených lidí, kde propojení hranou značí přátelství dvou lidí, sítě obchodních vztahů mezi společnostmi, neuronové sítě, metabolické sítě a spousty jiných. Dalším příkladem může být kolekce webových stránek, které se odkazují jedna na druhou [1, 10].

Ale proč bychom se měli zabývat sítěmi? Sítě jsou všude kolem nás. Čím dál tím víc systémů se dá modelovat pomocí sítí a jejich analýzou můžeme získávat zajímavé informace o reálném světě.

Problém je však v tom, že stávající sítě rostou a jejich analýza se stává obtížnější. V dnešní době máme však výpočetní prostředky, které nám pomáhají s analýzou rozsáhlých sítí. Úkolem je však vyvinout nástroje pro práci se sítěmi a pro jejich analýzu. Také chceme porozumět jejich struktuře a topologii a měřit jejich vlastnosti.

Speciální skupinou sítí jsou komplexní sítě, které mají tu vlastnost, že se dynamicky vyvíjejí v čase. U takových sítí je zajímavé zkoumat jejich vývoj a naučit se tento dynamický vývoj předvídat.

Studium grafů je jedním z pilířů diskrétní matematiky. Jako příklad využití teorie grafů se vždy uvádí problém sedmi mostů města Královce (Kaliningradu), který zdárně vyřešil v 18. století Euler. Dá se říci, že tento problém odstartoval rozvoj vědy teorie grafů.

2.1 Typy sítí

Skupina vrcholů spojených hranou je základní a nejjednodušší typ sítě. Existuje mnoho různých typů sítí a mohou být daleko složitější. Například v grafu může být více než jeden typ vrcholu nebo hrany. Každý vrchol nebo hrana může nésti určitý druh informace. Například v sociální síti lidí mohou vrcholy znázorňovat muže a ženy, lidi různých národností, věku, atd. Hrany mohou reprezentovat vztahy mezi těmito lidmi, přátelství, nepřátelství, profesní známosti, ale třeba i geografickou vzdálenost mezi lidmi. Hrany mohou být různé tloušťky, což může vyjadřovat, jak moc dobře jeden druhého zná [1].

Grafy mohou být orientované i neorientované. Orientované grafy mohou například vyjadřovat mobilní síť a posílání sms zpráv mezi mobily. Vrcholy by znázorňovaly mobilní telefony a hrany zprávy, které putují od jednoho přístroje k druhému – proto orientované grafy.

Existuje také typ hran - tzv. hyperhrany. Jsou to hrany, které mohou spojovat více než dva vrcholy. Jako příklad si můžeme uvést třeba síť lidí, kde hyperhranami jsou spojeni lidé, kteří patří do rodiny – rodinné vazby.

Dalším typem grafů mohou být například bipartitní grafy. Jsou to grafy, které obsahují dva odlišné typy vrcholů, kde hrany spojují pouze vrcholy navzájem různých typů.

Základní typy sítí:

1. Sociální sítě – živá společenství
2. Informační sítě – sítě webových stránek, citační sítě, blogy, ...
3. Technologické sítě – počítačové sítě, transportní sítě, ...
4. Biologické sítě – neuronové sítě, potravní řetězce, proteinové sítě, ...

2.2 Komunity

V sociálních sítích, které modelují například kooperaci lidí, lze pozorovat vytváření tzv. shluků (viz. obrázek č. 1). Shluky se myslí skupiny vrcholů, které mají vysokou hustotu hran v oněch skupinách, ale nižší hustotu mezi skupinami navzájem. Tyto shluky se odborně nazývají komunity. Jsou to tedy skupiny vrcholů, mezi kterými jsou spojení hustá, ale mezi jednotlivými komunitami jsou spojení řídká. Přesná definice komunit však neexistuje [1, 2].

Detekce komunit je snadná pouze u řídkých grafů. Řídké grafy jsou takové, které mají zhruba stejný počet hran jako vrcholů, tedy graf $G = (V, E), |V| = n, |E| = m$, je řídký, když $m = O(n)$ a hustý, když $m = O(n^2)$.

Komunity však nemusíme hledat jen v sociálních sítích. Lze je hledat například i v sítích informačních. Například v citačních sítích můžeme nacházet komunity, a to články, které se dělí do skupin podle tématu.

Metody pro nalezení komunit se dělí na tzv. globální nebo lokální. U globálních metod se předpokládá znalost celé datové kolekce. U velkých datových kolekcí to může znamenat, že kolekci nebudeme moci kvůli její velikosti zpracovat [8, 2].

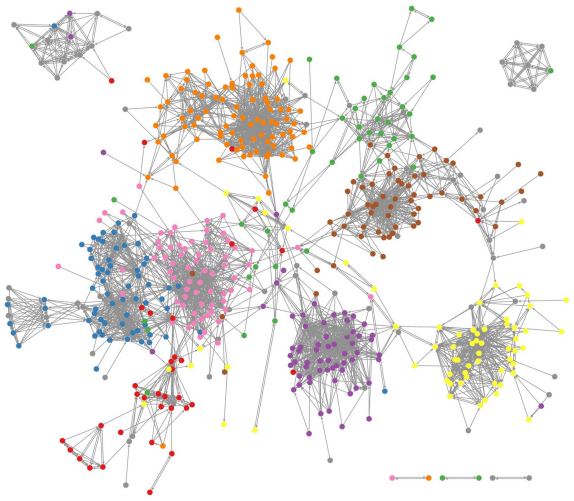
Naproti tomu u lokálních metod [12], celou strukturu sítě znát nemusíme. Stačí znát pouze strukturu části grafu. Lokální metody fungují tak, že si zvolíme libovolný vrchol nebo hranu grafu a řekneme o něm, že bude jádrem komunity. Postupně procházíme sousedy tohoto vrcholu a přidáváme je do komunity na základě určité metriky nebo heuristiky. Poté procházíme i sousedy vrcholů později přidaných do komunity.

Přesněji řečeno, pracujeme s hodnotou, která udává "kvalitu" komunity. Tuto kvalitu spočítáme před přidáním vrcholu do komunity i po přidání vrcholu do komunity. Když se po vložení vrcholu do komunity "kvalita" komunity zvětší, vrchol v komunitě ponecháme, jinak jej odstraníme.

Dalšími metodami pro detekci komunit je tzv. shluková analýza. Tato analýza funguje na takovém principu, že hledá shluky vrcholů v grafu a na základě určité podobnosti nebo vzdálenosti je seskupí do komunit. Podobností může býti míněno, že dva vrcholy jsou podobné tehdy, když sdílejí mnoho svých sousedů. Také můžeme definovat, že vrcholy patří do komunity, když jejich vzdálenost je menší než d_0 [3].

U algoritmu jednoduchého propojení se zjišťuje zda existuje posloupnost $x_1, ..., x_n$ taková, že každé dva po sobě jdoucí vrcholy jsou si podobné. Tyto vrcholy jsou pak v komunitě [3].

U algoritmu úplného propojení si musí být vrcholy v komunitě podobné navzájem všechny.



Obrázek 1: Vizualizace sítě, ve které jsou vidět shluky vrcholů [9]

2.3 Algoritmy pro layout

Se sítěmi a grafy se setkáváme v mnoha oborech. Lze pomocí nich modelovat různé situace. Data reprezentující rozsáhlé sítě je potřeba určitým způsobem interpretovat a vizualizovat. Tímto se zabývá disciplína zvaná „graph drawing“ – kreslení grafů. Existuje mnoho algoritmů pro layout a každý algoritmus se nám hodí v určité situaci. Některé dávají důraz na estetiku, jiné se zase zaměřují na rychlost a výkon vykreslení sítě. Některé vizualizují síť ve 2D prostoru, jiné jsou zase schopné vizualizace ve 3D. [4]

Před použitím kreslícího algoritmu je si nutné uvědomit, k čemu vlastně bude zobrazení sloužit a poté vybrat vhodný algoritmus pro layout. Existují algoritmy kreslící podle estetických kritérií, aby byly grafy co nejvíce čitelné a hezké na pohled.

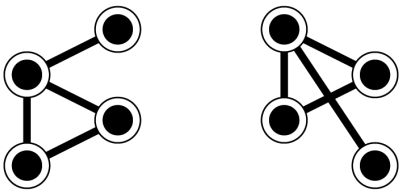
Pravidla pro estetičnost grafu (viz. obrázek č. 2):

1. Minimalizování hran, které se kříží.
2. Snaha o to, aby incidentní vrcholy měly stejnou vzdálenost jeden od druhého.
3. Nedovolit, aby vrcholy překrývaly hrany, které s nimi nejsou incidentní.

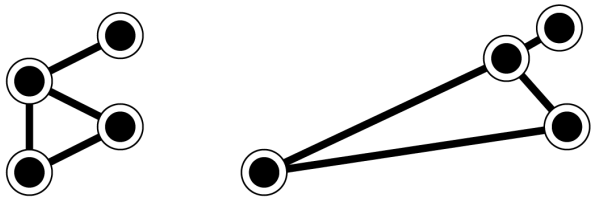
Je potřeba si ale uvědomit, že ne vždy je pěkný vzhled cestou, jak získat z grafu určité informace (viz. obrázek č. 3).

2.3.1 Kruhový layout

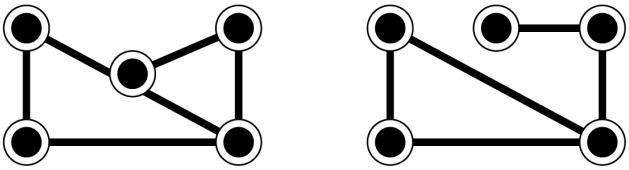
Jedním z algoritmů pro vizualizaci grafu je kruhový layout (viz. obrázek č. 5). Tento algoritmus funguje na takovém principu, že vykreslí vrcholy na pomyslnou kružnici a spojí jednotlivé vrcholy hranami [4, 5].



(a) Křížení hran



(b) Vzdálenost mezi vrcholy



(c) Překrývání hran a vrcholů

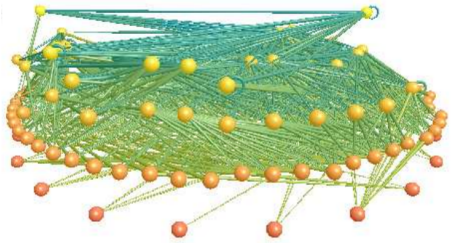
Obrázek 2: Estetická pravidla

Hrany mohou být rovné nebo mohou být kreslené formou křivky nebo oblouku. Kreslení rovných čar je méně náročné než vykreslování hran pomocí oblouků, ale snižuje se čitelnost grafu. Hrany kreslené formou oblouku jsou estetičtější, lépe se čtou, ale jsou výpočetně náročnější při výpočtu zaoblení hrany. Rádius oblouku je závislý na rozdílu kruhových pozic incidentních vrcholů grafu (spojené hranou). Výsledek vizualizace záleží na zvoleném pořadí vrcholů kolem dokola.

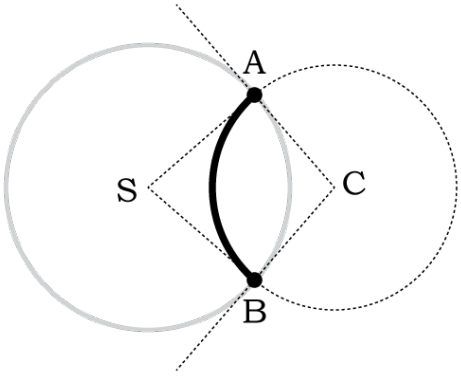
Kreslený oblouk je část kružnice, jejíž střed je v bodě C. Tento bod C je průnikem tečen na kružnici v bodě A a B (viz. obrázek č. 4).

Výhody:

- Kruhová pozice může reprezentovat určitou informaci, např.: věk, zeměpisnou šířku, atd.
- Je velmi rychlý – není potřeba nad sítí provádět žádné výpočty, vykreslí se vrcholy, a ty se poté spojí hranou.



Obrázek 3: Vizualizace, která není vhodná pro získávání informací z grafu [1]



Obrázek 4: Názorná ukázka způsobu vykreslení hrany jako oblouku

Nevýhody:

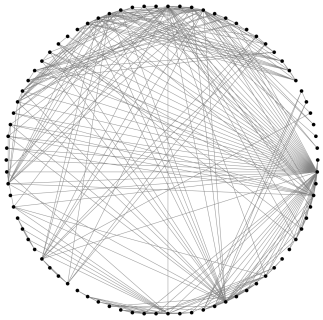
- Je těžké interpretovat rozsáhlou síť.
- Spousta křížících se hran – špatná čitelnost.
- Spousta dlouhých hran.

Mezi tyto algoritmy patří:

1. Circular layout
2. Radial Axis Layout

2.3.2 Arc diagram layout

Tento algoritmus slouží k vykreslení grafu podél pomyslné čáry. Hrany jsou kreslené formou kruhového oblouku. Tento typ algoritmu se používá, když vrcholy nesou dlouhé informace – když popisky vrcholů zabírají spoustu místa. Vrcholy mohou být seřazeny náhodně. Existují však techniky, které seřazením v určitém pořadí zmenší rozlehlost grafu

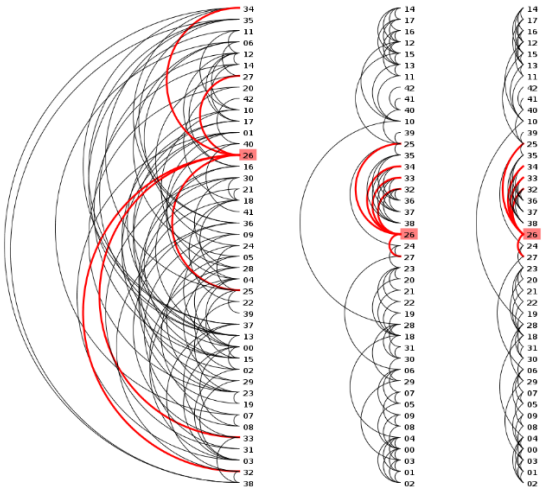


Obrázek 5: Vizualizace grafu kruhovým algoritmem

a to zmenšením velikosti oblouku. Tím se také zvýší čitelnost grafu a informace jsou snáze pochopitelné. Jedna z technik se nazývá tzv. „Barycenter heuristic“.

„Barycenter heuristic“ je iterativní technika, kde se počítá průměrná pozice sousedů každého vrcholu a poté se vrcholy setřídí podle této pozice. Tím se docílí přesunu vrcholů blíže ke svým sousedům a zapříčiní zkrácení oblouku. Ke snížení rozlehlosti grafu (do šířky) existuje technika, která oblouky nevykresluje s rozpětím $180°$, ale s menším úhlem oblouku (viz. obr. 6).

Nevýhodou tohoto typu algoritmu je, že není absolutně vhodný pro rozsáhlé sítě s mnoha hranami – snižuje se čitelnost grafu, zvyšuje se velikost grafu do šířky a s rostoucím počtem vrcholů i do výšky grafu.

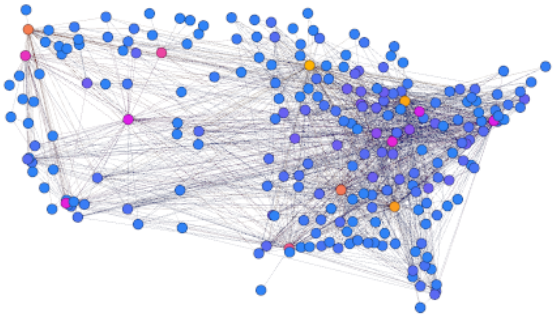


Obrázek 6: Vizualizace algoritmem Arc Diagram [4]

2.3.3 Geografické sítě a mapy

Tyto algoritmy používají, pro vizualizaci a umísťování vrcholů, souřadnice zeměpisné šířky a zeměpisné délky (viz. obrázek č. 7). Tento typ algoritmu využívá například aplikace

Google Map. Algoritmus je vhodný pro sítě s velkou mohutností, až milion vrcholů. Složitost algoritmu je $O\left(n^2\right)$ [5].



Obrázek 7: Vizualizace algoritmem pro Geolayout [14]

2.3.4 Force-directed algoritmy

Je to skupina algoritmů pro kreslení grafů, která se zaměřuje na estetiku vizualizace. Existují algoritmy pro kreslení jak ve 2D, tak ve 3D prostoru. Tedy úkolem této skupiny algoritmů je rozmístit vrcholy tak, aby se z nich informace co nejlépe četly – aby se v grafu křížilo co nejméně hran, nejlépe žádné, aby incidentní vrcholy byly od sebe zhruba stejně vzdálené, atd. Pozice jsou počítány na základě sil [4, 5].

V grafu jsou simulovány dvě síly a to odpuzování a přitahování. Tyto síly působí mezi vrcholy. Vrcholy se tedy navzájem odpuzují, ale incidentní vrcholy (spojené hranou) jsou u sebe drženy pomyslnou pružinkou, která jeden vrchol ke druhému přitahuje.

Algoritmus funguje následovně. Na počátku mají vrcholy náhodně vygenerovanou pozici. Algoritmus postupně přemisťuje vrcholy na základě sil působících mezi nimi, až do doby, kdy se jejich pozice určitým způsobem ustálí – rozdíl mezi odpudivou a přitažlivou silou je minimální (viz. obrázek č. 11).

Výhody:

- Kvalitní výsledky pro grafy střední velikosti (přibližně 100 vrcholů).
- Flexibilita – algoritmus je lehce modifikovatelný k přidání dodatečných kriterií pro zvýšení estetičnosti. To dělá tyto algoritmy univerzální.
- Algoritmy jsou jednoduché, je možné je naimplementovat několika řádky kódu.
- Jednou z výhod je interaktivní aspekt. Kreslením přechodných fází může uživatel vidět jak se graf vyvíjí a to ze změti chaoticky uspořádaných vrcholů po pěkně vypadající rozmístění.

Nevýhody:

- Dlouhá výpočetní doba – složitost algoritmu je $O\left(n^3\right)$, kde n je počet vrcholů. K výpočtu pozic je nutno n iterací – $O\left(n\right)$. V každé takovéto iteraci je ještě nutné vzít v úvahu všechny dvojice vrcholů a vypočítat nad nimi odpudivou a přitažlivou sílu $O\left(n \cdot n\right) = O\left(n^2\right)$. Z toho vyplývá výsledná složitost $O\left(n^3\right)$.

Mezi tyto algoritmy se řadí například algoritmy:

1. Fruchterman-Reingold – pomalý algoritmus, ale často používaný, vhodný pro zhruba 1.000 vrcholů, jeho složitost je $O\left(n^2\right)$.
2. Force atlas – algoritmus zaměřující se na kvalitu vizualizace, je vhodný pro grafy s až 10.000 vrcholy, jeho složitost je $O\left(n^2\right)$.
3. Force atlas 2 – je to vylepšená verze algoritmu Force atlas, navýšení počtu vrcholů, které je algoritmus schopný zpracovat až na 1.000.000 vrcholů, přitom si zachovává kvalitní výsledky vizualizace, nahrazuje odporovou a přitažlivou sílu tzv. „scaling“ parametrem.
4. YifunHa Multilevel layout – rychlý algoritmus s dobrými výsledky vizualizace, vhodný pro až 100.000 vrcholů, jeho složitost je $O\left(n \cdot log\left(n\right)\right)$.

2.4 Sammonova projekce

V praxi často potřebujeme pracovat s daty, jejichž definiční obor je více než třírozměrný. Při velkých definičních oborech se těžko nahlíží na strukturu těchto dat lidským okem. Je tedy potřeba určitým způsobem převést data do podoby, ve které se data budou snáze analyzovat a budou se dát snadno představit [6, 7].

Jednou z možností, jak toho dosáhnout, je tzv. Sammonova projekce. Je to nelineární projekce z n-rozměrného prostoru do m-rozměrného prostoru, kde n $\gg$ m, tak, aby byly co nejvíce zachovány vzdálenosti mezi vektory původního vysokodimenzionálního prostoru a vektory v prostoru s nižší dimenzí.

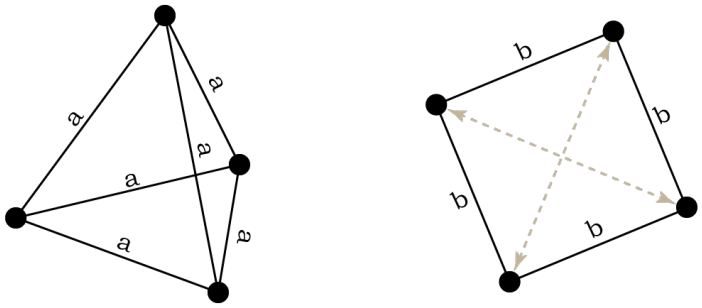
Se Sammonovou projekcí souvisí speciální funkce - tzv. ”hodnotící funkce”, která posuzuje ”kvalitu”projekce. Protože je funkce chybová, snažíme se tuto chybu minimalizovat, abychom dosáhli nejlepší možné projekce.

$$E = \frac{1}{\sum_{i<j}^{m}\left(d_{ij}^{*}\right)} \cdot \sum_{i<j}^{m} \frac{\left(d_{ij}^{*} - d_{ij}\right)^2}{d_{ij}^{*}} \tag{1}$$

E je chyba projekce (tzv. Sammon’s stress), d_{ij}^{*} je vzdálenost mezi vektory X_i a X_j z původního prostoru, d_{ij} je vzdálenost vektorů Y_i a Y_j v prostoru s nižší dimenzí, m značí počet vektorů.

Sammonova projekce nám sice pomáhá nahlédnout na data se složitější strukturou, ale má i svá omezení a nedostatky. Některé nedostatky demonstruje následující příklad (viz. obrázek č. 8).

Jako vstupní množina vektorů nám poslouží tzv. tetraedr, což je objekt, který má 4 vrcholy, každé 2 vrcholy jsou od sebe stejně vzdálené a každé 3 vrcholy tvoří stěnu objektu.



Obrázek 8: Ukázka použití Sammonovy projekce na tetraedr. Vlevo je objekt z třírozměrného prostoru, vpravo je transformovaný objekt do dvojrozměrného prostoru. Je zde znázorněn nedostatek převodu z třírozměrného prostoru do dvojrozměrného - na obrázku vpravo nejsou všechny vrcholy stejně vzdálené jeden od druhého (úhlopříčka). Čáry neznázorňují hrany grafu, ale vzdálenosti vrcholů jeden od druhého.

Množinu budeme promítat z prostoru o třech rozměrech do prostoru o dvou rozměrech.

Problém je tedy v tom, že ve 2D prostoru nemůžeme nalézt žádný objekt, který by splňoval všechny vlastnosti tetraedru. Například ve 2D prostoru nemůžeme najít takové čtyři body, které by od sebe byly stejně vzdálené, ale pouze tři.

Nemůžeme tedy zavést takovou projekci, ve které platí, že poměry vzdáleností z jednoho prostoru do druhého jsou totožné. Z této chyby také vyplývá, že při projekci nelze zaručit, že vektory, které jsou v jednom prostoru od sebe vzdálené nejdál, budou mít tuto vlastnost i v prostoru transformovaném.

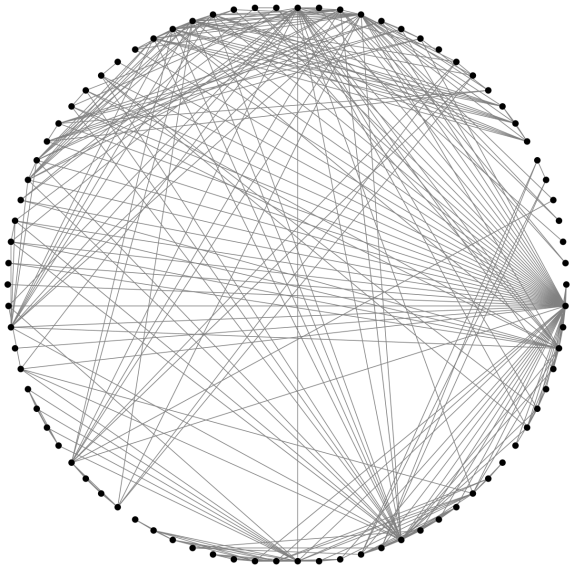
Touto metodou se také transformují data, která jsou v konečné fázi vizualizována aplikací, která je cílem této práce.

Sammonova projekce je tedy prostředkem, který nám dává možnost nahlédnout na multidimenzionální data a umožnit přibližnou představu o struktuře dat. Má však i své nedostatky, které plynou z nemožnosti zachovat všechny vlastnoti při projekci do nižších rozměrů, proto není možné se vždy spolehnout na zobrazenou strukturu dat.

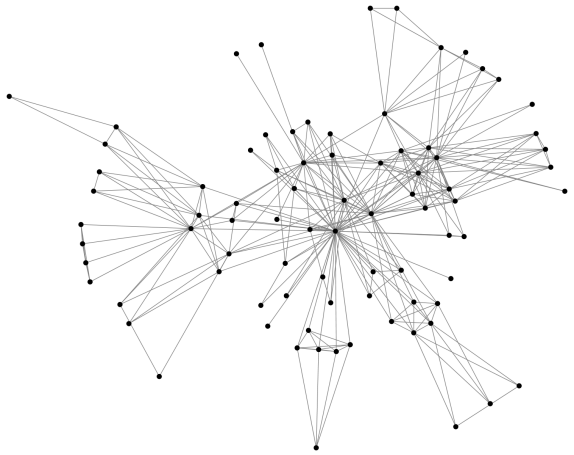
2.5 Aplikace pro layout

V této části si ukážeme ukázky vizualizace sítí na jedné datové množině. K tomu nám poslouží tři aplikace, které se tímto tématem zabývají.

K vizualizaci použijeme NodeXL[1], což je plugin pro Microsoft Excel. Dále použijeme programy Gephi[2] a Pajek[3]. Na obrázcích můžeme pozorovat přednosti i nedostatky jednotlivých algoritmů. Všechny vizualizují stejnou množinu dat (viz. obrázky č. 9, 10, 11, 12, 13, 14).

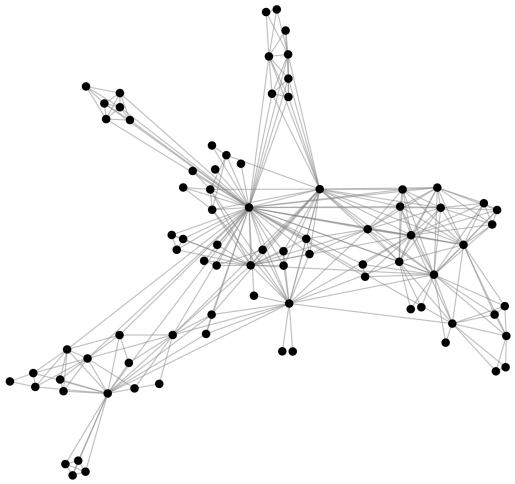


Obrázek 9: Vizualizace algoritmem Circular layout

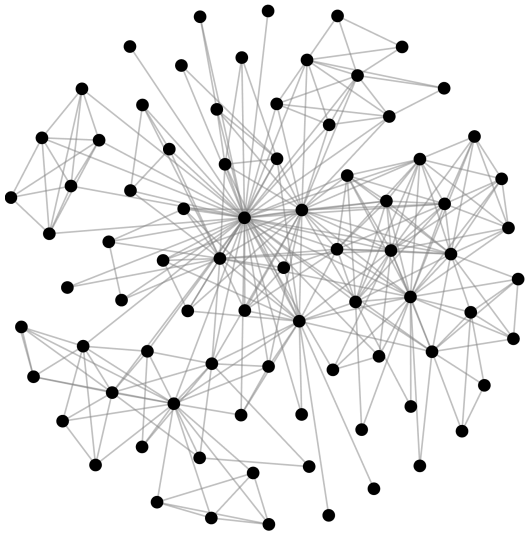


Obrázek 10: Vizualizace algoritmem Harel-Koren

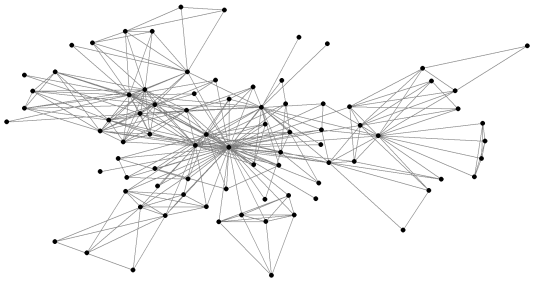
[1]http://nodexl.codeplex.com/
[2]http://gephi.org/
[3]http://pajek.imfm.si/doku.php?id=download



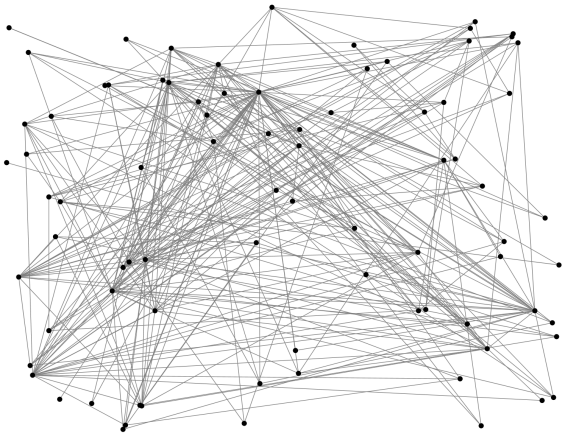
Obrázek 11: Vizualizace algoritmem Force Atlas



Obrázek 12: Vizualizace algoritmem Fruchterman-Reingold



Obrázek 13: Vizualizace algoritmem Kamada-Kawai



Obrázek 14: Vizualizace algoritmem pro náhodný layout

3 WebGL

3D grafika je jednou z důležitých disciplín už od počátků počítačů. Používá se například k vizualizaci dat, simulaci určitých dějů, vývoji počítačových her, atd. V dávnějších dobách si musel uživatel, který chtěl pracovat s určitou grafickou aplikací, tuto aplikaci obstarat, například si ji stáhnout přes internet, a nainstalovat. Tak to ostatně bylo se všemi aplikacemi, nejen s grafickými. V dnešní době se trendy výrazně změnily a vývoj aplikací se zaměřuje spíše na webovou platformu – aplikace pro internetový prohlížeč. Už dnes máme spousty webových aplikací, které lidé používají a přitom se nemusí zabývat jejich stahováním a instalací na operační systémy svých počítačů, ani stahováním pluginů do svých webových prohlížečů. [11]

Tato skutečnost určitě napomohla k myšlence zahrnout do webové platformy i možnost grafické vizualizace ve 3D. A proto se od roku 2012 pracuje na vývoji rozhraní pro kreslení 3D grafiky pro webovou platformu v rámci HTML5 pod názvem WebGL.

WebGL je nový standard pro 3D grafiku na webu. Má podporu v široké škále zařízení od počítačů přes tablety až po mobilní zařízení (je také nezávislý na operačním systému – multiplatformní). Výhodou této technologie je, že může naplno využít výkon zařízení, protože webový prohlížeč pracuje přímo s grafickým hardwarem zařízení (grafickou kartou), a to vše za použití javascriptu. Co se týče podpory webových prohlížečů, tak WebGL zatím nemá kompletní podporu u všech prohlížečů, a když, tak ne podporu celého programového rozhraní technologie. V době psaní tohoto textu mají téměř plnou podporu prohlížeče Google Chrome a MS Internet Explorer (Mozilla má podporu většiny příkazů, ne však všech).

WebGL je vyvíjeno a udržováno sdružením Khronos Group[4], jehož nejznámějšími členy jsou například Google, nVidia, Apple Inc., Oracle, a spousty dalších. Toto sdružení také spravuje standardy pro OpenGL (mulitplatformní grafická knihovna), ze které WebGL částečně vychází – konkrétně z OpenGL ES 2.0, což je verze OpenGL určená pro tzv. embedded systems (ES), tím je míněno pro použití v malých zařízeních jako jsou mobilní zařízení, tablety, herní konzole a PDA.

Jak to tedy vlastně všechno funguje? WebGL je javascriptové aplikační rozhraní. Toto rozhraní nám zpřístupňuje HTML element canvas, který je taktéž novinkou v HTML5. Jak už název napovídá – je to plátno, do kterého se kreslí. Rozhraní se zpřístupní metodou getContext, která jako parametr přebírá řetězec, který specifikuje, jaké rozhraní pro kreslení chceme používat (kromě WebGL umí canvas zpřístupnit i rozhraní pro kreslení 2D grafiky, které je oproti WebGL založeno na jednoduchém kreslení čar, obdélníků, textu, atd.). Pro zpřístupnění WebGL rozhraní se jako parametr vkládá řetězec „experimental-webgl“. Protože standard HTML5 ještě není oficiálně kompletně vytvořen, proto je přidán prefix „experimental-“. Následuje ukázka zdrojového kódu, který demonstruje zpřístupnění WebGL rozhraní (viz. zdrojový kód č. 1).

```
var canvas = document.getElementById("canvasElement"); // Ziskani elementu
var gl = canvas.getContext("experimental-webgl"); // Ziskani kontextu WebGL
```

Výpis 1: Získání kontextu pro 3D kreslení

[4] http://www.khronos.org/

Kreslení pomocí WebGL je složitý proces, který se skládá z mnoha kroků. Následuje jednoduchý seznam kroků, který je nutné provézt pro vykreslení určitého grafického primitiva.

1. Vytvořit element canvas
2. Získání webgl kontextu z elementu canvas
3. Nastavit vykreslovací rozmezí v canvasu (tzv. Viewport)
4. Vytvořit jeden nebo více bufferů
5. Vytvoření matic
6. Vytvoření shaderů
7. Nastavení shaderů a jeho parametrů
8. Vykreslení

Jak je psáno v předchozím odstavci, nejdříve je nutné vytvořit určitý prostředek, který bude vyrenderovanou grafiku zobrazovat. K tomu slouží HTML element canvas. Ten můžeme vytvořit staticky v HTML zdrojovém kódu nebo dynamicky v rámci kódu javascriptu.

Poté, co máme vytvořené plátno, do kterého se bude vykreslovat grafika, je potřeba získat určitý prostředek pro kreslení 3D grafiky. Ten získáme metodou elementu canvas, getContext (viz. zdrojový kód č. 1).

Jakmile máme získaný kontext WebGL u elementu canvas, musíme WebGL zadat obdelníkové hranice, kam máme v úmyslu do plátna kreslit. Ve WebGL je to nazváno viewport. Nastavení je jednoduché – stačí zavolat metodu *viewport* se čtyřmi parametry, které specifikují obdelníkové hranice (vzdálenost zleva, vzdálenost shora, šířka a výška). Následuje ukázka zdrojového kódu, který tyto hranice nastavuje (viz. zdrojový kód č. 2).

```
gl.viewport(0, 0, canvasElement.width, canvasElement.height);
```

Výpis 2: Nastavení hranic vykreslování

Nyní, když jsme WebGL sdělili, do jaké oblasti bude kresit, můžeme přejít k vytvoření bufferů. Co to vlastně buffery jsou? Buffery jsou pole hodnot, které obsahují pozice vrcholů, které budou kresleny. Buffery však nemusí obsahovat pouze pozice vrcholů, ale také mapovací souřadnice jednotlivých vrcholů, barvy vrcholů a nebo velmi důležité normálové vektory (používané při výpočtu osvětlení polygonů).

Důležité je vědět, že tyto buffery se přenáší přes WebGL rozhraní do grafického hardwaru zařízení, aby byl výkon co nejvyšší.

Základní objekty, se kterými WebGL umí pracovat se nazývají grafická primitiva a jsou to trojuhelníky, úsečky a body. Žádná jiná primitiva ve WebGL neexistují. Složitější tělesa jsou kreslena pomocí těchto nejjednodušších primitivních těles. Například obdélník vytvoříme pomocí dvou pravoúhlých trojuhelníků.

3.1 Matice

Matice jsou základním prostředkem, kterým se nastavuje kamera ve WebGL. Pohled kamery se skládá ze tří matic a jejich vzájemného složení. Jsou to matice projekční, pohledová a modelová.

Projekční matice slouží k zadání projekce. Typickými projekcemi jsou tzv. perspektivní a ortografická. Perspektivní projekcí vnímá například lidské oko. Vzdálenější objekty se jeví menší než objekty blízké bodu pohledu, i když mají stejnou velikost. Typickým příkladem jsou koleje, které se v dáli sbíhají v jednom bodě.

Jinou projekcí, využívanou v počítačové grafice, je ortografická projekce. Typickou vlastností je, že se vzdáleností se nemění velikost objektů. Využívají ji většinou strojaři při navrhování součástek, například v CAD programech.

Matice pohledová je další důležitá matice. Díky ní můžeme specifikovat pozici kamery v prostoru a její směr pohledu kudy se kamera dívá.

Poslední maticí je matice modelová. Ta slouží k modifikaci souřadného systému na základě souřadnic pozice kreslených objektů a jejich podobjektů.

Ve finále vše funguje tak, že se tyto tři matice mezi sebou vynásobí v pořadí projekční, pohledová a modelová. Výslednou maticí se poté transformují všechny vrcholy určitého objektu (přenásobením matice a vektoru – vektor zastupuje souřadnice vrcholu). Z takto transformovaného vektoru můžeme také zjistit pozici vrcholu na obrazovce (plátně – tedy převod z 3D do 2D) vzhledem k dané pozici kamery a jejího směru pohledu (viz. rovnice č. 14).

3.2 Shader

Shadery jsou programy, které se aplikují na určité entity, ke kterým se shader váže (např.: vrchol - vertex shader, pixel - fragment shader, atd.) a co je hlavní – vykonávají se na grafickém hardwaru zařízení, na kterém je aplikace spuštěna. Existuje několik druhů shaderů jako třeba vertex shader, fragment shader, geometry shader a tessellation shader. Ve WebGL však jsou jen dva druhy a to vertex shader a fragment shader.

Jazyk zdrojového kódu shaderu má stejnou syntaxi jako programovací jazyk C, rozšířený o několik klíčových slov. Také musí obsahovat jednu důležitou funkci s názvem “main”, která je hlavní funkcí programu, stejně jako programu napsaného v jazyce C++.

Zdrojový kód shaderu se musí před odesláním do grafické karty zkompilovat. Když kompilace nedopadne dobře – ve zdrojovém kódu shaderu je chyba, je možné si chybu získat a vypsat.

Vertex shader se aplikuje na všechny vrcholy, které se tomuto shaderu předají jako vstup. Má za úkol transformovat 3D souřadnici vrcholu na 2D souřadnici na plátně (obrazovce) (viz. zdrojový kód č. 3). Je v něm také možné počítat barvu při gouraudovém nebo konstantním stínování.

```
uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;
```

```
attribute  vec3 in_Vertex;

void main()
{
    gl_Position  = projectionMatrix  *  viewMatrix * modelMatrix * in_Vertex;
}
```

Výpis 3: Kód jednoduchého vertex shaderu

Fragment shader se aplikuje na všechny pixely kresleného a už zrasterizovaného primitiva (trojuhelníku, bodu, úsečky) a má za úkol nastavit barvu pixelu (viz. zdrojový kód č. 4). V tomto shaderu se také aplikují textury. Díky speciálním texturám a fragment shaderu můžeme také docílit speciálních efektů hrbolatosti aniž bychom museli zvýšit segmentaci objektu – tyto metody se nazývají normal mapping a bump mapping. Fragment shader se také používá pro výpočet barvy pixelu při Phongovém stínování.

```
uniform vec3 color;

void main()
{
    gl_FragColor = vec4(color,  1.0);
}
```

Výpis 4: Kód jednoduchého fragment shaderu

Poté, co máme vytvořené shadery, vytvořené buffery se souřadnicemi vrcholů objektů, vytvořené matice projekce a pohledu, nastavené hranice kreslící oblasti, už zbývá jen poslední krok a to svázání určitého shaderu s "WebGL vykreslovacím řetezcem" a nastavení jeho vstupů a parametrů.

Shadery mají tzv. uniformní proměnné a vstupní atributy. Vstupní atributy jsou proměnné, které nabývají hodnot obsažené v bufferech. Takže mohou nabývat například hodnot zastupujících souřadnice nebo barvy vrcholů.

Uniformní proměnné jsou takové proměnné, které během jednoho vykreslovacího cyklu nemění svou hodnotu. Můžou to být různé stavové hodnoty, konstanty specifické pro určitý vykreslovací cyklus. Matice kamery se také definují jako uniformní proměnné, protože během vykreslení jednoho objektu nepotřebujeme měnit jejich hodnotu.

Před nastavením uniformní proměnné si musíme získat odkaz na tuto proměnnou vyjádřený celočíselnou hodnotou generovanou WebGL. Ten získáme použitím funkce *getUniformLocation* a jako parametr dosadíme jméno proměnné umístěné ve zdrojovém kódu shaderu. Poté můžeme funkcí *uniform{1, 2, 3, 4}{f, i, fv}*[5] nebo *uniformMatrix{2, 3, 4}fv*[6] nastavit hodnotu proměnné.

U atributů je to podobné. Podle názvu proměnné si necháme vrátit adresu a přes tuto adresu nasměrovat proud hodnot z bufferů do shaderů. Ještě je potřeba funkcí *enableVertexAttribArray* aktivovat atribut v shaderu. Na závěr funkcí *drawArray* vykreslíme objekt.

[5] Např.: *uniform3f(...)* nebo *uniform1i(...)*
[6] Např.: *uniformMatrix4fv(...)*

4 Analýza a návrh aplikace

Tato kapitola je zaměřena na tvorbu aplikace z pohledu softwarového inženýra a stručně jsou popsány technologie, ve kterých je aplikace tvořena. Je rozebrán návrh tříd a je stručně popsáno, k čemu určitá třída slouží a jakou v aplikaci hraje roli. Dále jsou pak rozebrány vybrané a zajímavé problémy, které se objevily při implementaci. Také jsou zde popsány algoritmy, které jsou při tvorbě aplikace použity.

4.1 Požadavky na aplikaci

Aplikace bude sloužit k zobrazování sítí a jejich vlastností. Její hlavní doménou bude prostor o třech rozměrech, tedy 3D. Hlavní funkcí aplikace bude možnost zobrazení komunit. Vrcholy a hrany patřící do určité komunity budou muset být nějak odlišeny od ostatních vrcholů a hran, které do této komunity nepatří. Bude také možno si komunity zobrazit bez okolních vrcholů a hran, které do komunity nepatří – pouze vrcholy a hrany patřící do konkrétní komunity.

Další zobrazitelné vlastnosti budou váhy jednotlivých vrcholů a také hran. Bude však možno zobrazit síť i bez těchto vlastností. Hrany tedy budou moci být vážené i nevážené. Hrany také budou moci být úplně skryté.

Aplikace poběží na webové platformě. Cílem aplikace však není počítat souřadnice jednotlivých vrcholů. Tato data bude aplikace získávat z webové služby.

V aplikaci také bude potřeba implementovat vhodným způsobem výběr objektů – uživatel si bude moci označit vrcholy i hrany sítě a poté si třeba, na základě označení, zobrazit informace o označených objektech.

Uživatel také bude mít možnost jednotlivé vrcholy přesouvat poté, co jeden nebo skupinu z nich označí myší. Aplikace také bude poskytovat grafické rozhraní, které bude obsahovat informace o určitých objektech (vrcholech, hranách, komunitách).

Další drobnou funkcí je zobrazení popisků vrcholů, které si uživatel bude moci vypnout.

4.2 Použité technologie

WebGL: Aplikace bude vytvářena pro webovou platformu. Pro kreslení 3D grafiky byla zvolena technologie WebGL, která má výhodu, že není potřeba instalací dodatečných pluginů a zásuvných modulů pro webový prohlížeč a má podporu grafického hardwaru počítače (viz. http://www.khronos.org/webgl/).

TypeScript: Další zvolenou technologií byl zvolen programovací jazyk TypeScript a to ve verzi 0.95. Tato technologie byla zvolena, protože kompilátor TypeScriptového kódu převádí kód přímo do javascriptu. Jeho výhodou je, že TypeScript poskytuje typovou kontrolu oproti Javascriptu a přidává možnost se na kód dívat z pohledu objektově orientovaného programování, které zahrnuje vytváření tříd, rozhraní a dalších aspektů OOP. To vše urychluje vývoj aplikací a předchází tvorbě zbytečných chyb a jejich budoucímu hledání a řešení (viz. http://www.typescriptlang.org/).

HTML5 a CSS (+LESS): Jak už je psáno výše, aplikace bude vytvářena pro webovou platformu a její neodmyslitelnou součástí je technologie HTML. Bude také potřeba jednotlivým elementům nastavovat vizuální vlastnosti. K tomu poslouží technologie CSS. CSS zdrojový kód však bude kompilován z kódu technologie LESS, která je obdobou technologie CSS, ale je programátorsky přívětivější (viz. http://www.html5.cz/).

IDE: Aplikace bude tvořena v prostředí Microsoft Visual Studio 2012, protože technologie TypeScript lze doinstalovat pouze na MS Visual Studio 2012 nebo 2013 (viz. http://www.visualstudio.com/).

4.3 Použité knihovny

TSM: Knihovna, která v projektu bude použita, je knihovna TSM. Je to knihovna zaměřující se na matice, vektory a funkce pro jejich použití z hlediska 3D grafiky. Poskytuje například funkce pro tvorbu matic důležitých pro kameru – projekční matice (perspektivní a ortografickou) a pohledové matice. Dále knihovna obsahuje funkce pro výpočet vzdálenosti dvou bodů v prostoru, výpočty normálových vektorů, výpočet úhlu mezi dvěma vektory, transformace vektorů maticemi a spousty dalších. Tato knihovna je napsána v jazyce javascript, ale je k dispozici definiční soubor (soubor s rozhraními tříd) pro jazyk TypeScript (viz. https://github.com/vexator/TSM).

JQuery: Je to rychlá a na funkce bohatá javascriptová knihovna. Věnuje se manipulaci s HTML dokumentem, řízení událostí, animaci a tzv. AJAXu (viz. http://www.jquery.com).

4.4 Návrh tříd

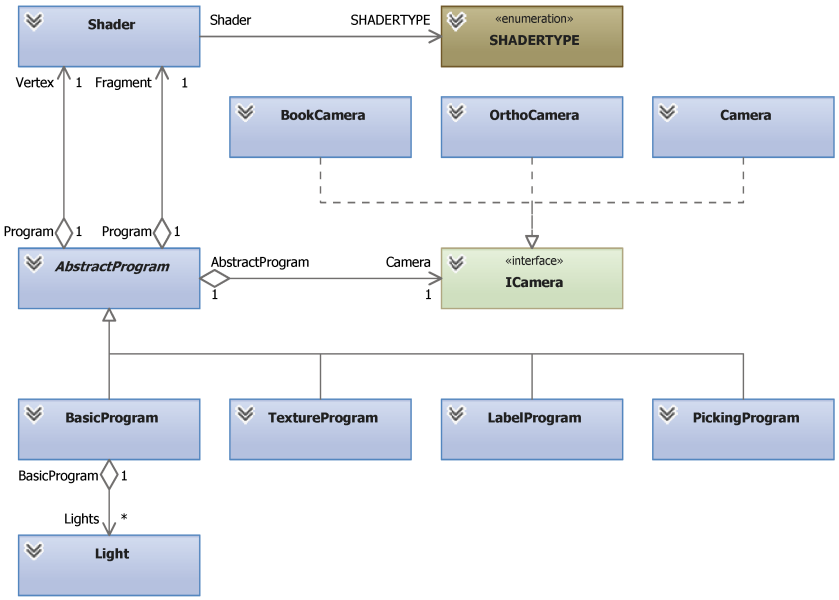
Na následujících obrázcích (15, 16, 17, 18) si můžeme prohlédnout diagramy tříd. Diagramy neobsahují všechny třídy, ale pouze jen ty nejdůležitější. Třídy jsou rozděleny do logických celků, které budou nyní popsány.

Diagram tříd na obrázku č. 15 ukazuje hierarchii shaderů a kamer. Důležitou třídou je *AbstractProgram*, která zapouzdřuje procedury WebGL. Účelem třídy *AbstractProgram* je renderování objektů třídy *MeshObject*. K tomu potřebuje mít definovanou kameru. Tu získává prostřednictvím rozhraní *ICamera*. Třída také vlastní dva shadery (Vertex shader a fragment shader).

Ze třídy *AbstractProgram* dědí čtyři třídy. Každá z těchto tříd se soustřeďuje na vykreslování něčeho určitého. *BasicProgram* je určený k renderingu objektů a používá Phongovo stínování, proto také vlastní pole světel, které jsou k tomuto stínování důležité. *LabelProgram* je určený k renderingu popisků vrcholů. *TextureProgram* je určený k renderingu textur a *PickingProgram* se zaměřuje na rendering obrazu, který slouží pro výběr objektů (viz. kapitola 5.1).

Třída *AbstractProgram* pro svůj chod potřebuje dva shadery. Fragment shader a vertex shader. Třídu pro tyto shadery reprezentuje třída *Shader*.

V aplikaci se používají různé druhy kamer. Jednu z nich reprezentuje třída *Camera*, která se používá pro pohyb ve scéně. Další kameru reprezentuje třída *BookCamera*. Ta se používá při renderingu knížky (viz. kapitola č. 5.3), která zobrazuje informace. Poslední kameru reprezentuje třída *OrthoCamera*, která se používá při renderingu popisků vrcholů. Všechny tyto třídy kamer implementují rozhraní *ICamera*. Tím se užití kamer stává obecnější.



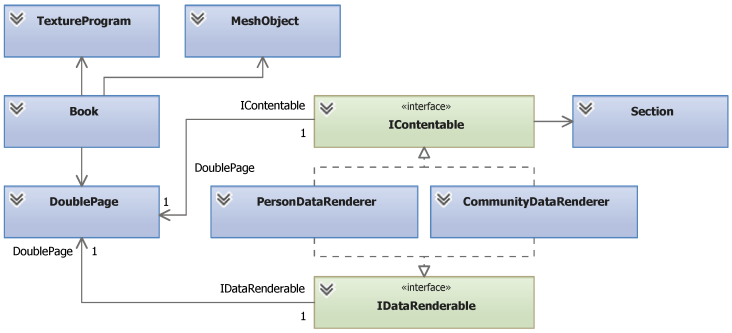
Obrázek 15: Diagram tříd (shadery, kamery)

V diagramu tříd na obrázku č. 16 můžeme vidět kooperaci skupiny tříd, které se zabývají vykreslováním knížky zobrazující různé informace. Samotnou knížku zobrazuje třída *Book*, která knihu a stránky vykresluje třídou *TextureProgram* a k popisu objektu používá třídu *MeshObject*.

Samotné informace pak vykreslují třídy *PersonDataRenderer*, která se zaměřuje na informace o jednom konkrétním vrcholu, a třída *CommunityDataRenderer*, která se zaměřuje na informace týkající se konkrétní komunity.

Tyto třídy po vykreslení vracejí pole objektů třídy *DoublePage*. Ta zastupuje dvojstranu z knížky. Tudíž obsahuje dvě textury (pro levou a pravou stránku), kde jsou vykreslené informace. Toto pole je pak předáváno třídě *Book*, která informace vykresluje podle toho, na které straně má uživatel nalistováno.

V třídním diagramu na obrázku č. 17 můžeme vidět třídy zaměřující se na popis grafové struktury a její samotnou vizualizaci. Samotný graf reprezentuje třída *Graph*. Ten se skládá z vrcholů a hran. Vrcholy reprezentuje třída *Vertex* a hrany třída *Edge*. Třída *Graph* také obsahuje informace o komunitách v grafu. Třída *Vertex* obsahuje metadata vrcholu, ty popisuje třída *VertexInfo*.



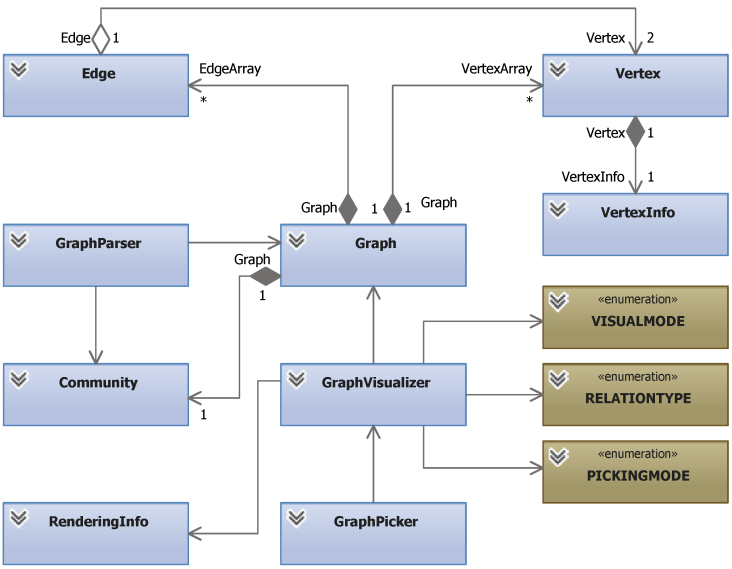
Obrázek 16: Diagram tříd (Knížka, DataRenderery)

Dekódování grafu z řetězce (získaného například přes webovou službu) zajišťuje třída *GraphParser*. Slouží i pro dekódování informací o komunitách z textového řetězce.

Pro samotné vykreslení grafu slouží třída *GraphVisualizer*. Přes tuto třídu se také nastavují různé vlastnosti zobrazení grafu. Například jakým způsobem se vykreslují hrany nebo zda se vykreslují vrcholy, které nepatří do aktuálně vykreslované komunity. Přes tuto třídu se také nastavuje aktuálně vykreslovaná komunita, atd.

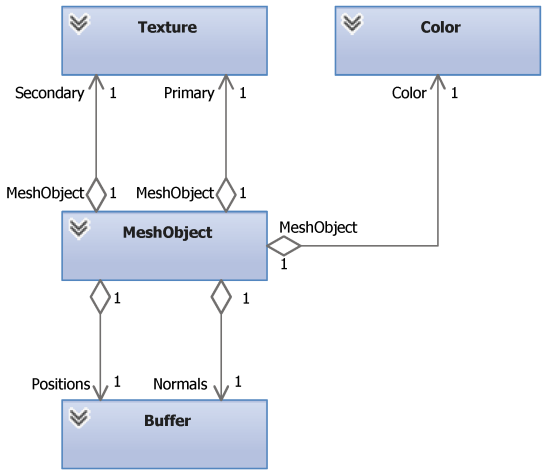
Třída *GraphPicker* slouží k vytváření obrazu, který se používá k detekci objektu, na který ukazuje kurzor. K tomu využívá objekt třídy *GraphVisualizer*, aby podle jeho vlastností věděl, které objekty vůbec mohou být vybrány.

S třídou *GraphVisualizer* souvisí třída *RenderingInfo*. Objekt této třídy se předává renderovacím metodám, aby věděly, které objekty mají určitým způsobem zvýraznit například proto, že jsou označené nebo na ně ukazuje kurzor myši.



Obrázek 17: Diagram tříd (třídy pracující s grafy)

V třídním diagramu na obrázku č. 18 můžeme vidět třídy zaměřující se na popis kreslených objektů. Hlavní třídou je třída *MeshObject*. Ta sdružuje všechny potřebné objekty pro popis určitého grafického objektu. Například obsahuje buffery reprezentující polygony nebo normálové vektory. Třída také obsahuje dvě textury - primární a sekundární (sekundární slouží pro multitexturing). Třída *Color* specifikuje barvu objektu.



Obrázek 18: Diagram tříd (Třídy pro popis objektu)

4.5 Sekvenční diagramy

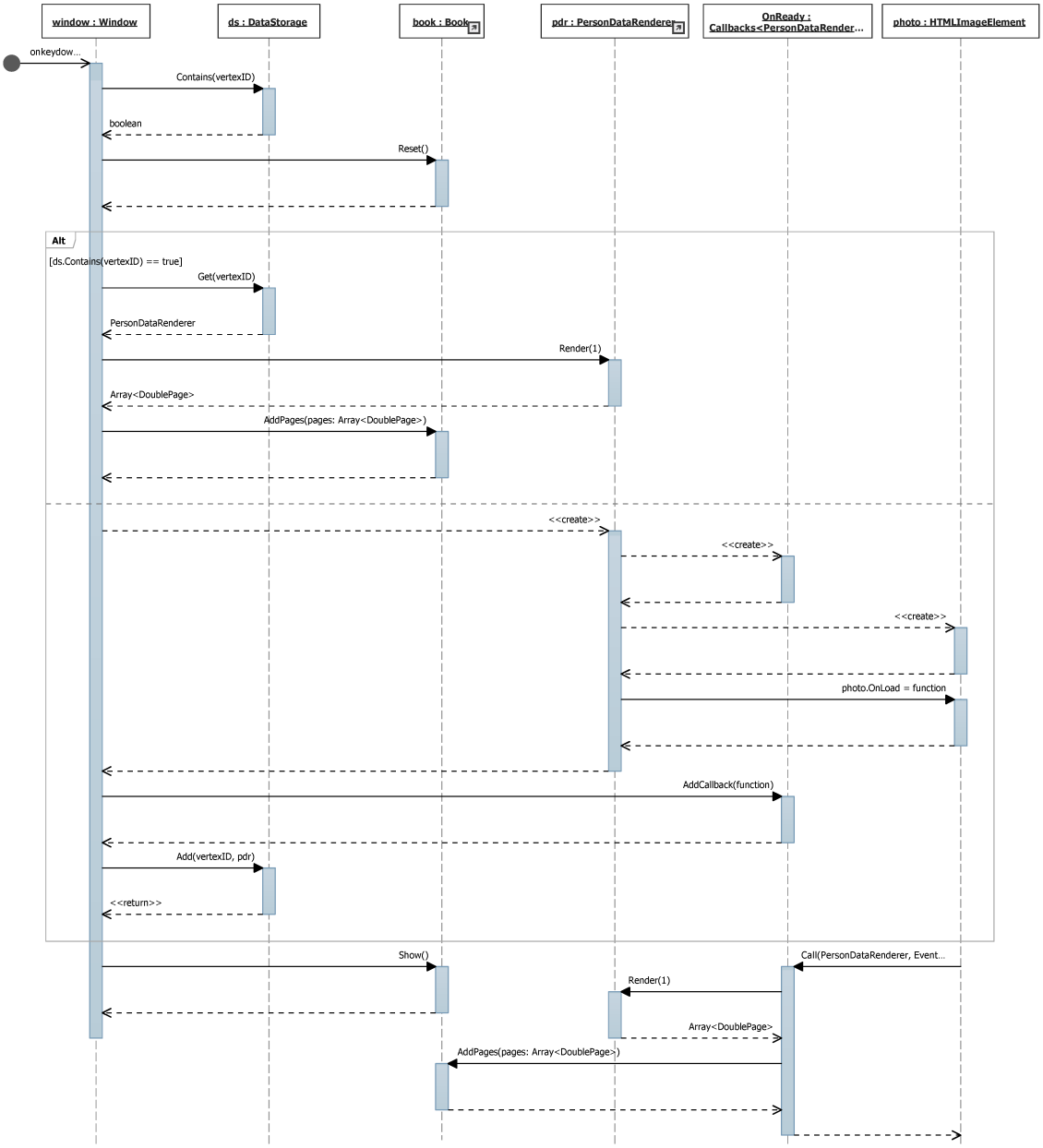
Na obrázku 19 je znázorněný sekvenční diagram, který popisuje operace, které předchází zobrazení knížky s informacemi. Využívá se zde návrhového vzoru lazy load, jehož cílem je načítat data, až když jsou potřeba.

Vše začíná stisknutím požadované klávesy a vyvoláním události *onkeydown* u objektu window. Tuto událost vyvolá prohlížeč. Poté se zjistí, jestli už neexistuje objekt, který se stará o vytvoření bitmap s informacemi (*PersonDataRenderer*).

Jestliže ano, objekt typu *PersonDataRenderer* získáme z objektu typu *DataStorage*. Objekt typu *PersonDataRenderer* nemusí znovu stahovat fotografie ze serveru, protože si je stáhl při minulém použití. Poté si z tohoto objektu necháme získat bitmapy v poli typu *DoublePage*, které předáme knížce.

Jestli se v objektu typu *DataStorage* nenalézá objekt, který se stará o vytváření bitmap (*PersonDataRenderer*), objekt se vytvoří a začne stahovat fotografii osoby, jejíž identifikátor je předán v konstruktoru (*id* osoby). Objekt se následně přidá do objektu *DataStorage*. Po načtení fotografie se vyvolá událost *OnReady* a může začít proces výroby bitmap - tedy stránek pro knížku. Poté jsou bitmapy předány knížce. Nakonec se metodou *Show* knížka zobrazí.

Na obrázku 20 je znázorněný sekvenční diagram, který popisuje načtení a parserování dat grafu vysláním požadavku na webovou službu.



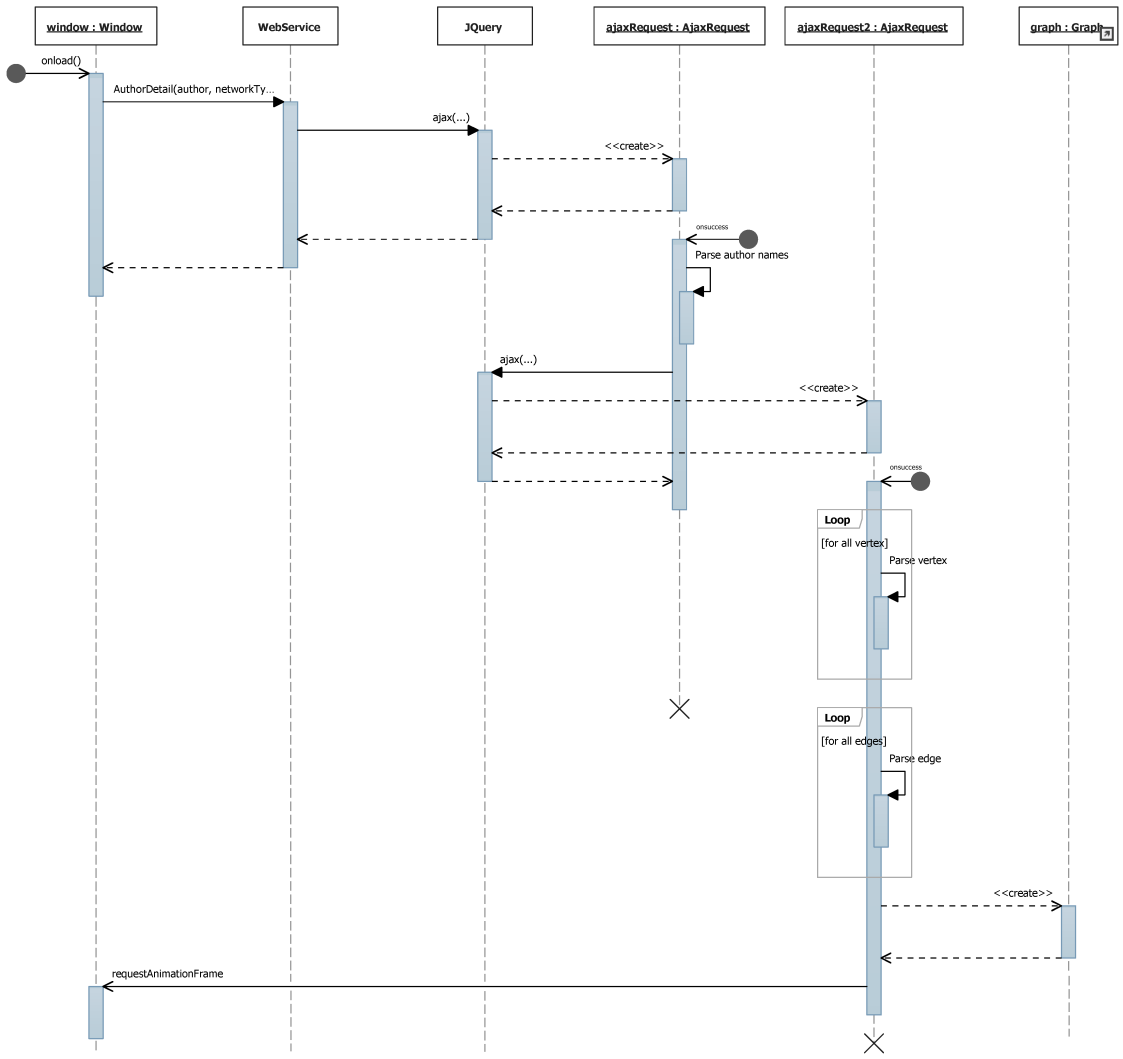
Obrázek 19: Sekvenční diagram - lazy load

Vše začíná zavoláním statické metody *AuthorDetail* na statickou třídu WebService. Tato metoda pomocí statické třídy *jQuery* a její statické metody *ajax* vyšle požadavek na webovou službu běžící na www.forcoa.net:9898. Požadavek vyvolá metodu webové služby *GetAuthorDetail* (viz. kapitola č. 5.7).

Až webová služba aplikaci odpoví, asynchronně se vyvolá událost onsuccess u objektu *ajaxRequest*. Následuje rozparserování odpovědi služby. V odpovědi jsou vráceny

jména autorů. Na základě těchto jmen se poté vytvoří další požadavek k získání 3D souřadnic vrcholů (autorů) grafu. Vyšle se požadavek pro zavolání metody webové služby *GetAuthorsLayout*.

Až webová služba aplikaci odpoví, asynchronně se vyvolá událost onsuccess u objektu *ajaxRequest2*. Metoda vrátí 3D souřadnice vrcholů. Poté vytvoříme objekt grafu, dále pak vytvoříme vrcholy a hrany na základě přijatých odpovědí na volané metody a předáme je objektu grafu. Na konec zavoláme anynchronní metodu *requestAnimationFrame* na objekt *window*, která spustí vykreslovací smyčku aplikace.



Obrázek 20: Sekvenční diagram - načítání a parserování dat grafu

5 Implementace

V této kapitole budou popsána některá zajímavá řešení implementace aplikace.

5.1 Výběr objektů

Jedna z důležitých funkcí každé grafické aplikace je možnost si myší kliknout na určitý objekt a s tímto objektem poté provádět další operace. WebGL přímo nepodporuje žádný způsob jak docílit identifikace objektu, na který uživatel klikl a nebo nad kterým objektem je v dané chvíli kurzor myši. Existuje však několik technik, jak této identifikace docílíme.

Ranné verze knihovny OpenGL, ze které WebGL vychází, poskytovaly možnost určitým způsobem identifikovat objekt pod kurzorem. Vše fungovalo na tak, že se pomocí speciální matice (tzv. picking matrix) změnila projekce kamery, že kamera zabírala oblast pouze několika pixelů v oblasti kurzoru (viz. obrázek č. 21). Následně se vykreslily všechny objekty. Poté se ze speciálního bufferu, kam se zaznamenávaly indexy všech objektů, které se vykreslily do malé oblasti zabírané kamerou, dalo zjistit, který objekt je pod kurzorem. Toto je však zastaralá technika a WebGL ji nepodporuje.

V moderním OpenGL se používá technika spojená s tzv. stencil bufferem. Je to speciální buffer, reprezentující dvourozměrné pole velikosti plátna, do kterého kreslíme. Ukládají se do něj celočíselné hodnoty, které si programátor před vykreslením nadefinuje. Hodnota se uloží do buněk v dvourozměrném poli, na pozicích pixelů zrasterizovaného primitiva. Do stencil bufferu by se tedy zapisovaly indexy objektů a to na pozice kreslených polygonů. Po vykreslení kompletní scény je potřeba si načíst hodnotu ze stencil bufferu na pozici kurzoru. A jelikož hodnoty ve stencil bufferu reprezentují indexy objektů v poli, tak tedy víme, na který objekt ukazuje kurzor (viz. obrázek č. 22).

Stencil buffer však zatím není ve WebGL kompletně implementovaný a proto se tímto způsobem nedá výběr objektů zrealizovat.

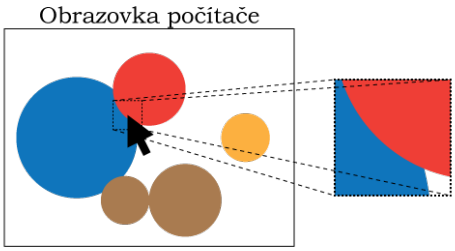
Existuje ještě technika, která příliš nezávisí na implementaci WebGL. Předpokladem je pouze schopnost umět zjistit barvu pixelu na dané pozici z frame bufferu. Toto je i technika, která byla při implementaci aplikace použita.

Vše funguje tak, že se každý objekt vykreslí určitou barvou. Poté se zjistí jakou barvu má pixel, který je pod kurzorem. Z této barvy je možno zjistit index vykresleného objektu, na který ukazuje kurzor (viz. obrázek č. 24).

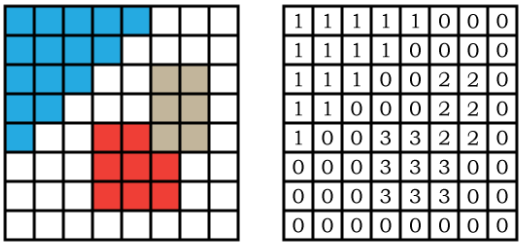
Při implementaci se však objevil problém. Při zapnutém antialiasingu, což je vyhlazovací technika, se pixely na okrajích objektů určitým způsobem zprůměrovávají. To zapříčiní, že se ze zprůměrované barvy nezíská správný index objektu, který je pod kurzorem.

Problém by se dal vyřešit vypnutím vyhlazování, ale výsledky obrazu by pak nebyly moc estetické – byl by příliš patrný rastr. Ještě by někoho mohlo napadnout vypínat vyhlazování jen při renderování pro výběr a zapínat vyhlazování při kreslení konečného obrazu. Problém je v tom, že vyhlazování se nedá dynamicky zapínat a vypínat.

Tento problém byl tedy vyřešen tak, že se nezjistí barva pouze jednoho pixelu, ale hned několika. Konkrétně celé mřížky pixelů např.: 3x3. Poté se tato mřížka pixelů projde a zaznamená se četnost všech druhů barev, které jsou v mřížce obsaženy. Poté se určí



Obrázek 21: Ukázka způsobu výběru za pomocí tzv. výběrové (picking) matice



Obrázek 22: Výběr objektů implementovaný s využitím stencil bufferu

barva pod kurzorem tak, že se z mřížky vezme barva s největší četností (viz. obrázek č. 23).

Může se zdát, že z hlediska výkonu tento způsob není úplně optimální, ale testy mluví jinak. Byla testována i mřížka velikosti 9x9 a aplikace byla stále plynulá.

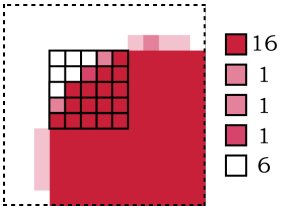
Následuje skupina vzorců, které byly navrženy pro zakódování indexu objektu do barvy pixelu. Výsledná barva se skládá ze 3 složek. Červená, zelená a modrá. Viz. rovnice č. 2.

$$color = (R, G, B) \tag{2}$$

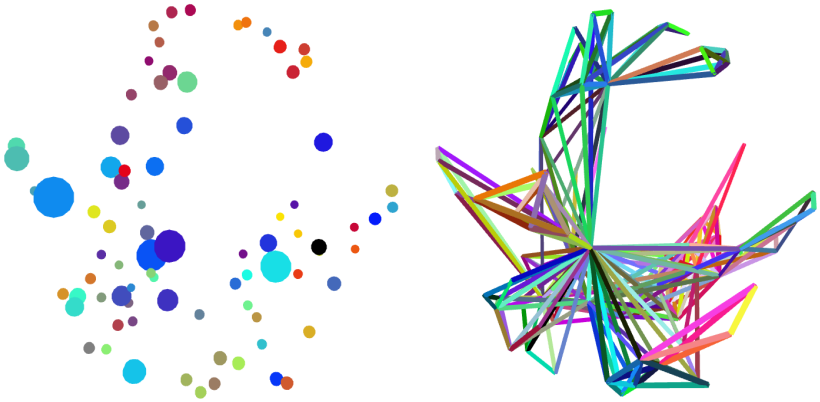
Následujícími vzorci se index objektu zakóduje do jednotlivých složek barvy. Funkce *floor* ořezává část čísla za desetinou čárkou. Proměnná *index* je číslo, které kódujeme. Proměnná *objectCount* vyjadřuje počet objektů v poli, které vykreslujeme. Viz. rovnice 3, 4, 5.

$$R = floor\left(\frac{index \cdot \frac{255^3}{objectCount}}{255^2} \ \ mod \ \ 255\right) \tag{3}$$

$$G = floor\left(\frac{index \cdot \frac{255^3}{objectCount}}{255} \ \ mod \ \ 255\right) \tag{4}$$



Obrázek 23: Obrázek ilustrující výběr analýzou mřížky barev. Sloupeček vpravo ukazuje četnost jednotlivých barev v mřížce. Z nějčetnější barvy se následně bude dekódovat index objektu.



Obrázek 24: Výběr prováděný technikou color picking

$$B = floor\left(index \cdot \frac{255^3}{objectCount} \mod 255\right) \tag{5}$$

Následující vzorec slouží k dekódování indexu objektu z barvy pixelu. Funkce *round* zaokrouhluje číslo na celé číslo. Proměnné *R*, *G*, *B* jsou jednotlivé složky barvy. Proměnná *objectCount* vyjadřuje počet objektů v poli, které byly vykresleny (viz. rovnice 6).

$$index = round\left(\frac{R \cdot 255^2 + G \cdot 255 + B}{255^3} \cdot objectCount\right) \tag{6}$$

5.2 Vizualizace objektů

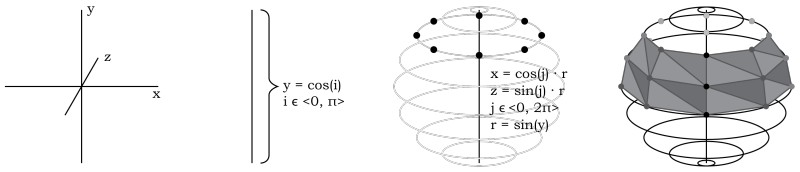
Tato část se bude věnovat implementaci vizualizace sítě. Bude zmíněno, které objekty pro vizualizaci byly použity a jak jsou tyto objekty vytvářeny. Také bude uvedeno, jaký stínovací model byl použit pro stínování objektů.

Graf se skládá z vrcholů a hran. Bylo rozhodnuto, že vrcholy budou vizualizovány formou kuliček a hrany formou dutých válců.

Typy objektů pro vizualizaci vrcholů a hran hrály roli při rozhodování, zda v projektu bude použit nějaký framework či knihovna pro práci s grafickými objekty. U složitějších a nepravidelných objektů se těžko tvoří geometrie objektů bez speciálního editoru. Framework by sloužil k importu této geometrie ze souboru. Ale tím, že v grafu budou figurovat pouze pravidelné objekty jako koule a válce, tak bylo rozhodnuto, že se žádný framework ani knihovna nepoužije. Pro vygenerování geometrie koule a válce existují jednoduché algoritmy.

5.2.1 Generování geometrie koule

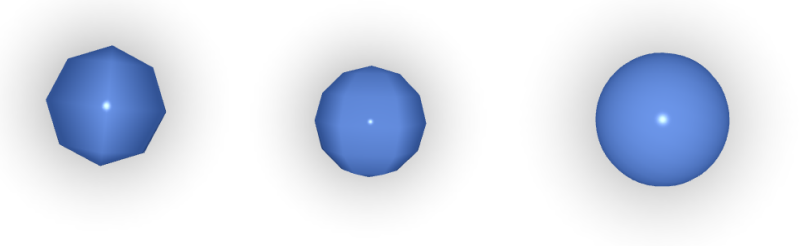
Generování geometrie se skládá ze dvou fází. První fáze je generování souřadnic vrcholů koule a druhou fází je pak tvorba polygonů z vygenerovaných vrcholů. První fáze programového kódu se tedy skládá ze dvou cyklů, kde první cyklus je určený ke generování y-nové složky souřadnice a druhý cyklus se stará o generování x-ových a z-ových složek souřadnice. Dá se říci, že se vrcholy nanáší na kružnici ve vrstvách postupně zeshora koule směrem dolů. Vše je názorně ukázáno na obrázku č. 25.



Obrázek 25: Generování geometrie koule

Druhá fáze je vytváření polygonů. Ty se tvoří takovým způsobem, že se vezmou nějaké dvě po sobě jdoucí vrstvy vrcholů (v kružnici) a vytvoří se trojúhelníky spojením vrcholů pod sebou (viz. obrázek č. 25).

Tohoto způsobu generování koule se v aplikaci využívá tak, že vrcholy grafu vykreslované blízko ohnisku kamery se vizualizují pomocí objektů s velkou segmentací, aby se zajistil dojem co možná nejvyšší hladkosti. Vrcholy grafu vykreslované dál od bodu pohledu se vykreslují se segmentací menší. Tím se zvyšuje rychlost vykreslování (viz. obrázek č. 26).



Obrázek 26: Vyrenderované koule reprezentující vrcholy grafu s různou segmentací

5.2.2 Stínovací model

Základní typy stínování jsou stínování konstantní, Gouraudovo a Phongovo. Při implementaci bylo rozhodováno, který z nich se použije. Rozhodováno bylo mezi Gouraudovým a Phongovým stínováním. Nakonec bylo rozhodnuto, že se použije Phongovo stínování, protože tímto stínováním se dosáhne estetičtějších výsledků, i když na úkor výkonu, protože Phongovo stínování je o trochu náročnější na výpočet než stínování Gouraudovo.

Výsledná barva je založená na Phongově osvětlovacím modelu. Barva se skládá z tzv. ambientní složky. Tato složka vyjadřuje okolní světlo, které není přímo vyzařováno ze žádných světelných zdrojů. Toto světlo vzniká tak, že se světlo z určitého zdroje několikrát odrazí (například od stěn v místnosti), a není tedy jasně patrný směr, odkud přichází [13].

Další složkou je složka difúzní. Tato složka vytváří trojrozměrný vzhled objektu. Tato složka se od objektu odráží do všech směrů, ale množství odraženého světla závisí na směru dopadu světla. Čím menší úhel je mezi normálovým vektorem a vektorem směru dopadu světla, tím je množství odraženého světla větší [13].

Třetí složkou je tzv. složka odrazová. Je to ta část světla, která dopadá na těleso ze světelného zdroje a odráží se do okolí podle zákona odrazu. Intenzita této složky závisí na vzájemné poloze světla, objektu a kamery [13].

Výsledná barva je tedy spočtena na základě intenzit jednotlivých složek světla (viz. rovnice č. 7 a 8).

$$color = I_a + I_d + I_s \tag{7}$$

$$color = I_a + I_L \cdot (N \cdot L) + I_L \cdot (R \cdot V)^h \tag{8}$$

I_a – intenzita ambientní složky, I_d – intenzita difúzní složky, I_s – intenzita odrazové složky, I_L – intenzita světla, N – normálový vektor, L – jednotkový vektor paprsku světla, R – jednotkový vektor a symetrický s vektorem L podle vektoru N, V – jednotkový vektor pohledu

Důležitým prvkem každého polygonu je normálový vektor. Tento vektor slouží k výpočtu osvětlení polygonu. Protože bylo použito Phongovo stínování, tak se stínování počítá pro každý pixel zrasterizovaného primitiva zvlášť. Interpolace probíhá pouze u normálových vektorů. Kdyby bylo použito stínování Gouraudovo, interpolace by probíhala nad barvami vrcholů, což by nedávalo příliš estetické výsledky.

Následuje zdrojový kód fragment shaderu, který počítá barvu pixelu podle Phongova osvětlení (viz. zdrojový kód č. 5).

```
void main()
{
    vec3 cameraRay = normalize(ex_cameraPosition − ex_Position);
    gl_FragColor = vec4(0.0, 0.0, 0.0, 0.0);

    for(int i = 0; i < MAX_LIGHTS; i++)
    {
        if ( lights [ i ]. enabled == 1)
```

```
        {
            vec3 lightRay = normalize(lights[i].position - ex_Position);
            float intensity = max(dot(ex_Normal, lightRay), 0.0);
            vec3 refl = normalize(reflect(-lightRay, ex_Normal));
            float spec = max(dot(refl, cameraRay), 0.0);
            gl_FragColor += 0.5 * pow(spec, 100.0) * vec4(lights[i].color, 1.0)
                + vec4(mix(color, lights[i].color, 0.5) * (intensity * 0.35), 1.0);
        }
    }
    gl_FragColor += vec4(0.65 * color, 1.0);
}
```

Výpis 5: Fragment shader počítající Phongovo osvětlení

Důležitou funkcí, která byla při výpočtu světla použita je skalární součin (viz. rovnice č. 9 a 10). Tato funkce vrací kosinus úhlu mezi dvěma vektory, které se předají jako parametr. Tím, že funkce vrací kosinus úhlu, se může intenzita barvy prostě přenásobit výsledkem této funkce a intenzita barvy pixelu se nastaví podle směru dopadu světla na plochu. Když se vektor dopadu rovná normálovému vektoru, intenzita je v tomto bodě nejvyšší, funkce vrátí 1. Když je vektor dopadu paprsku světla kolmý na normálový vektor, tak funkce vrátí 0. Tím se intenzita světla sníží na minimum, tedy na černou.

$$\cos(\alpha) = \frac{u \cdot v}{|u| \cdot |v|} \tag{9}$$

$$\cos(\alpha) = u \cdot v; |u| = 1, |v| = 1 \tag{10}$$

5.3 Zobrazení informací

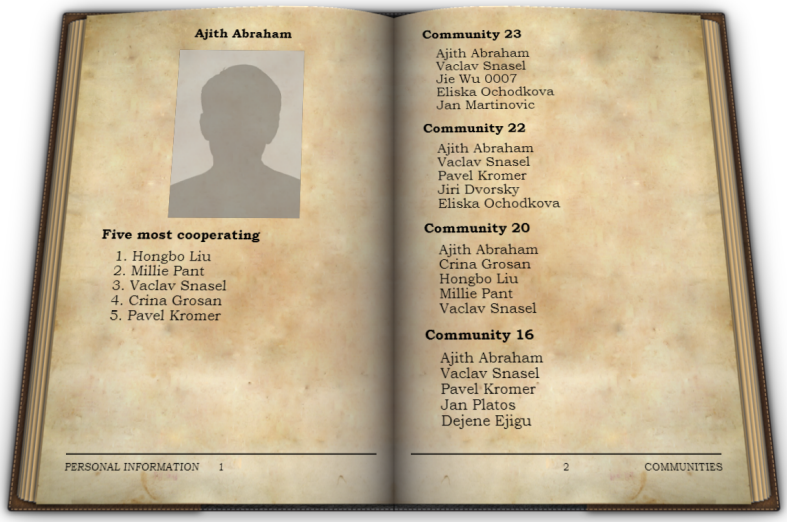
Jednotlivé objekty grafu (vrcholy a hrany) nesou informace. Tyto informace bylo potřeba určitým způsobem zobrazit. Bylo uvažováno mezi klasickým způsobem pomocí HTML elementů a nebo zaexperimentovat pomocí WebGL. Byla zvolena vizualizace pomocí WebGL.

Objektem pro zobrazení je knížka. V knížce je možné pomocí myši listovat. Tato knížka se zobrazí po označení některého z objektů grafu a stisknutí konkrétní klávesy (viz. obrázek č. 27).

Knížka je navržena tak, že na svých stránkách umí zobrazit jakoukoliv grafiku ve formě bitmapy. Takže informace zobrazené knížkou je potřeba vykreslit do bitmapy a tuto bitmapu poté předat knížce.

Knížce jsou bitmapy předány v poli objektů třídy *DoublePage*, která reprezentuje dvojlist knížky. Objekty této třídy pak knížka mapuje na svoje stránky.

Informace jsou renderovány do bitmap třídami *PersonDataRenderer* a *CommunityDataRenderer*. Třída *PersonDataRenderer* vykresluje informace o vrchole (tedy osobě). První stránka obsahuje jméno, příjmení, fotku dané osoby a až pět lidí, se kterými dotyčný nejvíce spolupracuje. Další stránky obsahují soupis komunit, do kterých daná osoba patří, společně s ostatními lidmi, kteří patří do těchto komunit.



Obrázek 27: Knížka zobrazující informace o osobě

Třída *CommunityDataRenderer* vykresluje informace o komunitě. Konkrétně vypisuje seznam lidí, kteří do této komunity patří, společně s jejich fotkami.

Tyto třídy informace vykreslují pomocí HTML elementu *Canvas* a rozhraní *CanvasRenderingContext2D*, které element implementuje. Tento element je poté předán WebGL. WebGL si poté převede data z elementu do grafického hardwaru počítače.

V jeden okamžik knížka vykresluje pouze dvě nebo tři stránky, aby se neplýtvalo výkonem na stránky, které stejně zrovna nejsou vidět. Knížka si drží ukazatel na aktuální stránku, kterou uživatel zrovna listuje. Knížka tedy vykresluje onu aktuálně listovanou stránku a jednu předcházející stránku a následující stránku. Když stránkou uživatel zrovna nelistuje, jsou vykreslovány pouze dvě stránky. Když je aktuální stránka otočená vlevo, tak se vykreslí aktuální a následující stránka. Když je aktuální stránka otočená vpravo, tak se vykreslí aktuální a předchozí stránka.

Aby se fotografie osob nemusely ze serveru stahovat při každém požadavku uživatele - zobrazit informace o osobě nebo komunitě, využívá se zde návrhového vzoru lazy load. Při požadavku zobrazit informaci se nejdříve zjistí, zda už informace někdy byly zobrazeny, tudíž, jestli už se jednou vytvořil *DataRenderer* a ten si stáhl fotografie osob. Když už se tak jednou stalo, použije se již vytvořený objekt, který už má všechny fotografie stáhnuté. Objekty se ukládájí do kolekce typu *DataStorage*. Tato třída funguje na principu klíč-hodnota.

Při renderingu stránek se využilo techniky tzv. multitexturingu. Je to technika vykreslování, kdy se na jeden polygon nanáší víc než jedna textura. Jednu texturu v tomto případě reprezentuje stránka knížky a druhou texturu reprezentuje bitmapa s vykreslenými informacemi (textem, obrázky, atd.).

Následující vzorec ukazuje, jakým způsobem jsou míchány barvy textur při multitexturingu (viz. rovnice č. 11).

$$color\,(r,g,b,a) = (mix\,(c1_{rgb}, c2_{rgb}, c2_a)\,, c1_a + c2_a)\,; \tag{11}$$

Výsledek je barva složená ze čtyř složek (r, g, b, a). Proměnná c_1, c_2 jsou míchané barvy. Funkce mix(C, D, E) je určená k míchání barev C a D podle parametru E. Parametr E je v intervalu $\langle 0, 1\rangle$. Když je parametr E roven 0.25, barva se namíchá ze 75% barvy C a 25% barvy D.

5.4 Popisky vrcholů

Tradiční aplikace pro vizualizaci grafů poskytují funkci nechat si zobrazit popisky vrcholů. Tato funkce tudíž nesměla chybět ani v této aplikaci.

Popisky vrcholů jsou zobrazovány pomocí technologie WebGL. Po načtení grafu jsou vygenerovány bitmapy, do nichž jsou vykresleny popisky vrcholů. Bitmapy jsou poté předány WebGL a poslány do grafické karty.

Jak ale zjistit, kam popisek umístit? Je potřeba zjistit souřadnici na plátně, na kterou se vrchol promítl z 3D prostoru do 2D prostoru. K tomu je potřeba znát vlastnosti kamery, kterou byla scéna vykreslena. Konkrétně projekční, pohledovou a modelovou matici.

Polohové vektory, před transformací maticemi, musejí být převedeny z kartézkého souřadného systému do homogenního souřadného systému. Matice jsou také v homogenním souřadném systému. Díky homogennímu systému můžeme popsat projektivní prostor. Matice tedy jsou velikosti 4×4. A tranformace vektoru se třemi složkami s maticí velikosti 4×4 není definováno. Proto je potřeba vektor převést do homogenního souřadného systému (rovnice č.12) rozšířením vektoru o jednu dimenzi. Homogenní souřadnici volíme nejčastěji rovno jedné.

$$A\,[x,y,z] \rightarrow A_h\,[x \cdot w, y \cdot w, z \cdot w, w] \tag{12}$$

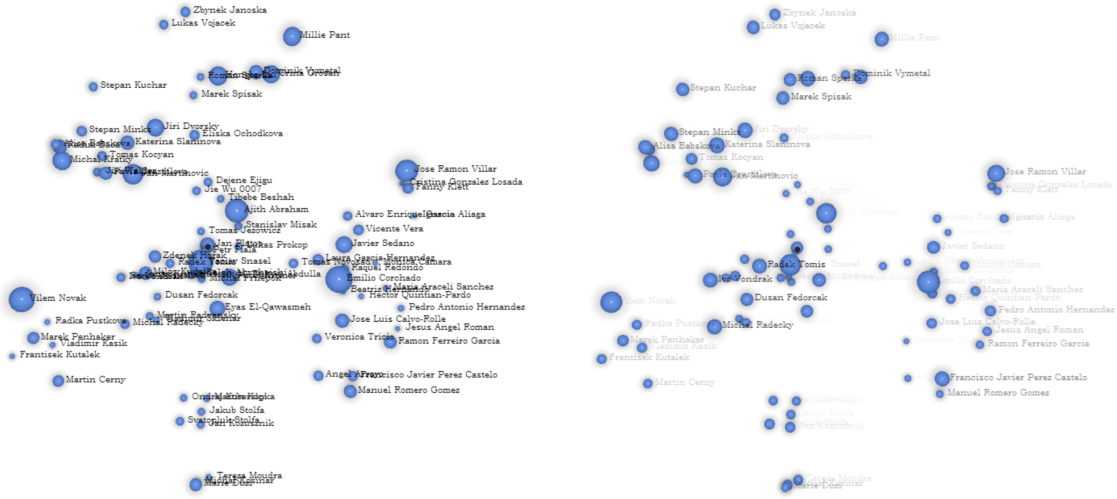
Když se součinem matic přetransformuje polohový vektor vrcholu, tak dostaneme mezivýsledek, u něhož složky vektorů, které se promítnou na plátno, jsou v intervalu $\langle -1; 1\rangle$ (rovnice č.13). Ty vektory, které se nepromítnou na plátno, mají hodnotu jednotlivých složek mimo tento interval. Poté je potřeba mezivýsledek převést na souřadnice pixelů obrazovky. K tomu slouží vzorec č.14.

$$u = (P \cdot V \cdot M) \cdot v \tag{13}$$

$$coords\,(x,y) = \left(\frac{\frac{u_x}{u_w} + 1}{2} \cdot width, \frac{1 - \frac{u_y}{u_w}}{2} \cdot height \right) \tag{14}$$

Nyní, když známe souřadnice vrcholů po promítnutí na obrazovku, můžeme vyrenderovat textury s popisky vrcholů. K tomu byla použita ortografická projekce.

U trojrozměrných grafů však může nastat problém, že se popisky můžou překrývat (viz. obrázek č. 28) a jsou pro uživatele hůře čitelné.



Obrázek 28: Popisky grafů

Proto bylo navrhnuto řešení, které spočívá v tom, že blízké vrcholy k bodu pohledu mají popisky výrazné oproti vrcholům vzdálenějším, které mají intenzitu popisku nižší nebo nulovou (viz. obrázek č. 28).

Před vykreslením popisků je nalezen nejbližší a nejvzdálenější vrchol od bodu pohledu. Intenzita popisku vrcholu je pak vypočtena na základě poměru jeho vzdálenosti od kamery a vzdálenosti nejbližšího a nejvzdálenějšího vrcholu. Následuje vzorec pro výpočet intenzity popisku (viz. rovnice č. 15).

$$intensity = \left(1 - \frac{distance - nearest}{farest - nearest}\right)^2 \tag{15}$$

5.5 Záře

Jedním z grafických prvků je dojem záře kuličky reprezentující vrchol grafu. Tento grafický prvek sám o sobě nevizualizuje žádnou informaci. Je zde zakomponován jen pro estetický dojem (viz. obrázek č. 29).

Dojem záře je vytvářen texturou, která je namapována na čtverec umístěný na pozici kuličky. Textura se generuje při startu aplikace.

Následuje zdrojový kód, generující texturu záře (viz. zdrojový kód č. 6).

```
var arr: Uint8Array = new Uint8Array(64 * 64 * 4);

for (var y: number = 0; y < 64; y++)
{
    for (var x: number = 0; x < 64; x++)
    {
        arr[y * 64 * 4 + x * 4 + 0] = R;
```

```
            arr[y * 64 * 4 + x * 4 + 1] = G;
            arr[y * 64 * 4 + x * 4 + 2] = B;

            var distanceFromCenter = Math.sqrt(Math.pow(x - 32, 2) + Math.pow(y - 32, 2)) / 32;

            if (distanceFromCenter > 1.0)
                distanceFromCenter = 1.0;

            arr[y * 64 * 4 + x * 4 + 3] = Math.pow(1.0 - distanceFromCenter, 2) * 255;
        }
    }
```

Výpis 6: Generování textury záře kuličky

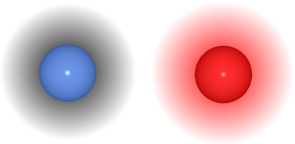
Čtverec je v prostoru natočený vždy směrem ke kameře. Přesněji řečeno, normálový vektor polygonu směřuje vždy do kamery. Natočení směrem ke kameře je prováděno tak, že je spočten úhel mezi vektorem směru z vrcholu do kamery, jehož y-ová složka je vynulována, a vektorem $[0, 0, -1]$ (viz. zdrojový kód č. 7). O tento úhel se poté čtverec orotuje kolem osy Y. Dále se spočítá úhel mezi vektorem směru z vrcholu do kamery a stejným vektorem až na y-ovou složku, která je rovna 0. O tento úhel pak čtverec rotuje kolem osy X. Poté normálový vektor čtverce směřuje do bodu pohledu.

```
var direction : TSM.vec3 = new TSM.vec3([dst.x - src.x, dst.y - src.y, dst.z - src.z]);
var dirZ: TSM.vec3 = new TSM.vec3([0.0, 0.0, -1.0]);

var tempVec: TSM.vec3 = new TSM.vec3([direction.x, 0, direction.z]);

// Y rotation
var cosAngleY: number = TSM.vec3.dot(tempVec.normalize(), dirZ);

var angleY: number = Math.acos(cosAngleY);
if (direction.x > 0)
    angleY = Math.PI * 2 - angleY;

// X rotation
var cosAngleX: number = TSM.vec3.dot(tempVec.normalize(), direction.normalize());

var angleX: number = Math.acos(cosAngleX);
if (direction.y < 0)
    angleX = Math.PI * 2 - angleX;

return new TSM.vec2([angleX, angleY]);
```

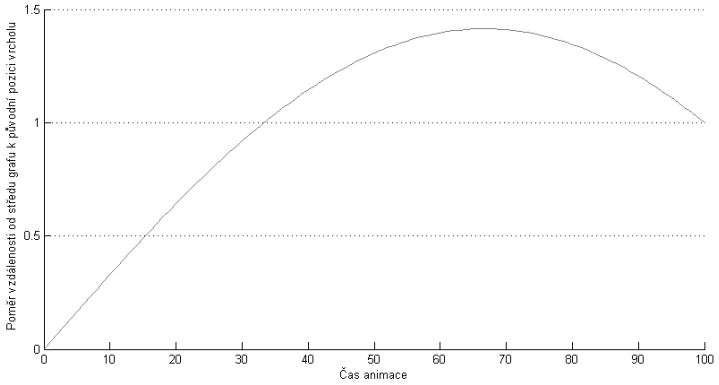
Výpis 7: Vypocet rotace polygonu

5.6 Efekt rozkvétající květiny

Jeden z pozdějších požadavků bylo přidání animace grafu po jeho načtení. Animace grafu dává dojem rozkvétající květiny. Vše probíhá tak, že všechny vrcholy vytrysknou z pomyslného středu grafu směrem ke svým původním pozicím. Vrcholy postupně zpomalují.



Obrázek 29: Záře vrcholů



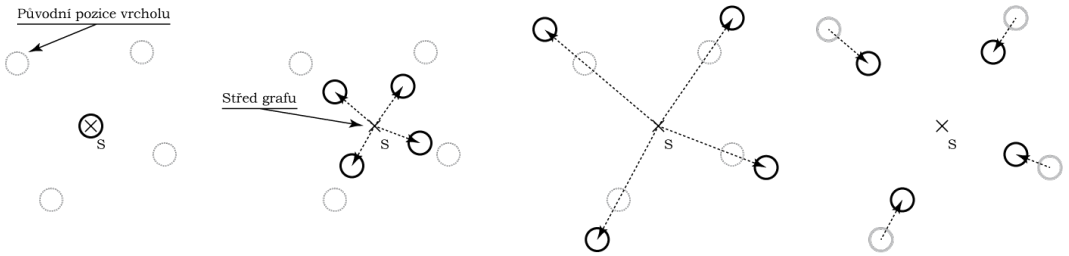
Obrázek 30: Průběh hodnoty poměru vzdálenosti od středu grafu k původní pozici vrcholu v čase

Vrchol se na trajektorii od středu grafu směrem k původní pozici zastaví kousek za původní pozicí. Poté začne vrchol padat zpět směrem ke středu. Vrcholy se při pádu zpět zastaví, až dosáhnou své původní pozice (viz. obrázek č. 31).

Hledání středu grafu se provádí tak, že se postupně procházejí všechny vrcholy a z jejich souřadnic se vyberou minimální a maximální hodnoty jednotlivých složek vektoru (souřadnice). U těchto složek se pak provede průměr a vznikne nová souřadnice, která reprezentuje střed grafu.

K animaci vrcholů slouží třída *GraphAnimator*. V konstruktoru se objektu předá graf, který obsahuje jednotlivé vrcholy. Třída pro animaci si uloží původní pozice jednotlivých vrcholů. Třída *GraphAnimator* obsahuje metodu *Update*, která při každém zavolání změní pozici vrcholů grafu předaného v konstruktoru.

Nová pozice se vypočítá na základě souřadnice středu grafu, původní pozice vrcholu a času animace. Hodnota času je při startu animace nastavena na hodnotu 0. Při zavolání metody *Update* se hodnota času zvýší. Na základě hodnoty času se vypočítá vzdálenost vrcholu od středu grafu. Pro výpočet vzdálenosti využívám goniometrické funkce *sinus(x)*, kde $x \in \left\langle 0; \frac{3}{4}\pi \right\rangle$. Touto funkcí se docílí, že se vrcholy postupně zpomalují a následně změní směr letu. Hodnota, kterou funkce *sinus* vrátí, je poté přenásobena $\sqrt{2}$. Tímto způsobem se docílí mírného zapružení vrcholu a vrácení se zpět na původní pozici. Průběh vzdálenosti je možné vidět na obrázku č. 30.



Obrázek 31: Animace grafu

5.7 Propojení s webovou službou

Cílem této práce je naimplementovat aplikaci takovým způsobem, aby byla částečně propojená s webovou službou a byla schopna přes tuto službu získávat potřebná data k vizualizaci sítě. Webová služba běží na serveru s url www.forcoa.net:9898. K této službě je možné se připojit pouze po připojení do školní sítě VŠB. Webová služba vrací odpovědi ve formátu JSON.

Aplikace nyní využívá dvě funkce webové služby. Těmi jsou *GetAuthorInfo* a *GetAuthorsLayout*. Následuje jejich stručný popis.

První z těchto metod vrací kompletní informace o autorovi v daném čase a zahrnuje informace o autorově "okolí". Okolím jsou myšleni autoři, kteří spolu v daném čase spolupracují (jsou spojeni hranou).

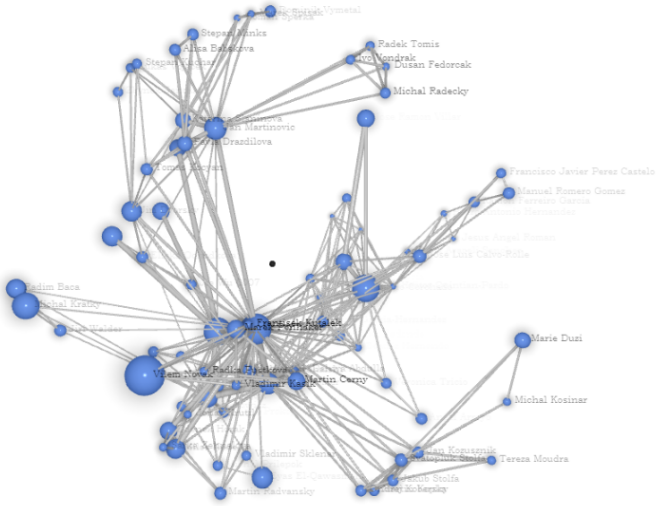
Druhá funkce je využívána pro získání 3D souřadnic vrcholů určité sítě autorů. Jako layoutovací algoritmus se zde využívá Sammonovy projekce (viz. kapitola 2.4). Kompletní definice těchto a dalších metod lze nalézt v dokumentaci webové služby.

Proces získání dat z webové služby začíná zadáním url do webového prohlížeče ve správném formátu. V url nesmí chybět parametry *name*, který udává jméno autora, jehož síť si chceme nechat zobrazit, dále pak parametr *depth*, který udává hloubku spoluautorské sítě a parametr time, který udává čas.

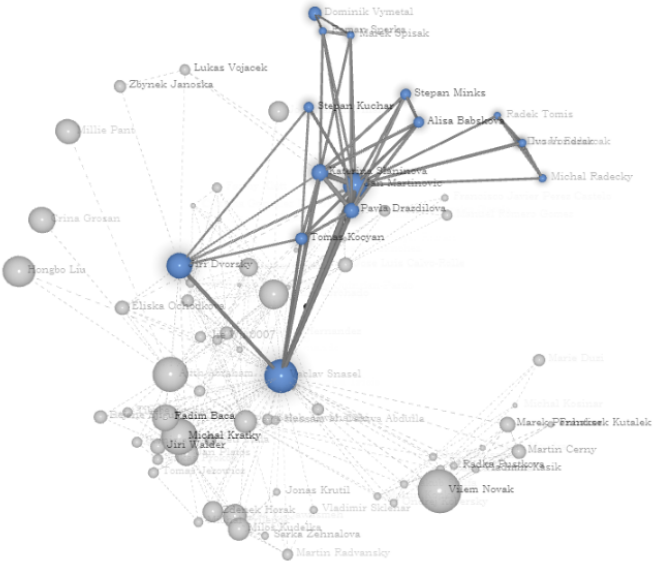
Aplikace si na základě těchto parametrů, napojením na webovou službu, získá potřebné informace o síti. Nejdříve se pomocí metody *GetAuthorInfo* získají informace o hlavním autorovi a autorech, se kterými hlavní autor v daném čase spolupracoval. Na základě těchto informací si pomocí funkce *GetAuthorsLayout* získáme 3D souřadnice těchto autorů.

5.8 Obrázky z aplikace

V této kapitole můžeme najít tzv. screenshoty výsledné aplikace. Je zde screenshot, který ukazuje základní vizualizaci sítě (viz. obrázek č. 32) a screenshot, na kterém můžeme vidět vizualizaci komunit (viz. obrázek. č. 33).



Obrázek 32: Ukázka výsledné vizualizace vytvořenou aplikací



Obrázek 33: Ukázka vizualizace komunit vytvořenou aplikací

6 Závěr

Hlavním cílem této práce bylo naimplementovat aplikaci, která by sloužila pro vizualizaci sítí ve 3D. Aplikace měla být naimplementována pro webovou platformu s využitím aplikačního rozhraní WebGL, získávat data pomocí webové služby, na kterou se aplikace měla být schopna napojit, dále pak vizualizovat vybrané vlastnosti sítí, především komunity, a také měla umožnit zobrazení informací o jednotlivých vrcholech a komunitách sítě. Dalším cílem bylo seznámit se s aplikacemi, které jsou zaměřené na vizualizaci sítí a s algoritmy pro layout, které tyto aplikace pro vizualizaci používají.

Byla naimplementována aplikace pro vizualizaci sítí a jejich vlastností. Aplikace byla naimplementována pro webovou platformu s využitím aplikačního rozhraní WebGL. Uživatel si v aplikaci může pomocí myši se sítí otáčet, posunovat ji a libovolně přibližovat či oddalovat kameru od sítě.

Hlavní funkce aplikace:

- Do aplikace byla naimplementována vizualizace komunit sítě.
- Uživatel si může zvolit, jakým způsobem se mají vizualizovat hrany sítě.
- V aplikaci je naimplementována identifikace objektu, na který ukazuje kurzor myši.
- Po výběru vrcholu si uživatel může nechat zobrazit informace o vybraném vrcholu.
- Aplikace také umožňuje zobrazení popisků vrcholů.
- Aplikace je schopna se napojit na webovou službu www.forcoa.net a získávat data pro vizualizaci sítí.

Cíle bylo dosaženo. V budoucnu by se aplikace určitě mohla rozšířit o další funkce. Například vylepšit komponentu knížky, aby v ní bylo možno vybírat prvky myší. Dále by pak v knížce mohl být zařazen obsah, ve kterém by si mohl uživatel zvolit, na kterou kapitolu či stránku chce nalistovat.

7 Reference

[1] Newman, M. E. J. *The Structure and Function of Complex Networks* [online], Department of Physics, University of Michigan, Ann Arbor, MI 48109, U.S.A. and Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, U.S.A., 2003, [cit. únor 2014]. Dostupné na: http://arxiv.org/pdf/cond-mat/0303516.pdf.

[2] Ochodková, E. *Grafové algoritmy - Hledání komunit* [online], VŠB – Technická univerzita Ostrava, 2013. [cit. březen 2014]. Dostupné na: http://www.cs.vsb.cz/ochodkova/courses/gra/gal10_2013.pdf.

[3] Ochodková, E. *Grafové algoritmy - Hledání komunit 2* [online], VŠB – Technická univerzita Ostrava, 2013. [citováno v březnu 2014], Dostupné na: http://www.cs.vsb.cz/ochodkova/courses/gra/gal11_2013.pdf.

[4] McGuffin, M. J. *Simple Algorithm for Network Visualization: A Tutorial* [online], Department of Software and IT Engineering, Ecole de technologie superieure, Montreal, H3C 1K3, Canada. [cit. březen 2014]. Dostupné na: http://profs.etsmtl.ca/mmcguffin/research/2012-mcguffin-simpleNetVis/mcguffin-2012-simpleNetVis.pdf.

[5] Gephi Tutorial Layouts [online] [citováno v březnu 2014]. Dostupné na: http://www.slideshare.net/gephi/gephi-tutorial-layouts.

[6] Petr Hummel, *DIPLOMOVÁ PRÁCE*, ČVUT Praha - Fakulta Elektrotechnická, 2001.

[7] Radvanský, M. – Kudělka, M. – Horák, Z. – Snášel, V. *Network Layout Visualization Based on Sammon's Projection* [online], VŠB – Technická univerzita Ostrava, 2013. [cit. březen 2014].

[8] Freiherr, J. *Identifakce lokálních komunit v sociálních sítích*, Bakalářská práce, VŠB – Technická univerzita Ostrava, 2013.

[9] Fortunato, S. *Community detection in graphs* [online], Complex Networks and Systems Lagrange Laboratory, ISI Foundation, Viale S. Severo 65, 10133, Torino, I-ITALY, 2010. [cit. březen 2014]. Dostupné na: http://arxiv.org/pdf/0906.0612.pdf.

[10] Ochodková, E. *Grafové algoritmy a komplexní sítě* [online], VŠB – Technická univerzita Ostrava, 2013. [citováno v březnu 2014]. Dostupné na: http://www.cs.vsb.cz/ochodkova/courses/gra/gal1_2013.pdf.

[11] Parisi, T. *WebGL: Up and Running* [online], O'reilly, 2012. [citováno v březnu 2014]. Dostupné na: http://it-ebooks.info/book/918/.

[12] Bagrow, J.P. – Bollt, E.M. *Local method for detecting communities*, Phys. Rev. E 72 (4) (2005) 046108

[13] Tišnovský, P. *Grafická knihovna OpenGL (18): vykreslování osvětlených těles* [online], ROOT.CZ [citováno 5.4.2014]. Dostupné na: *http://www.root.cz/clanky/opengl-18-vykreslovani-osvetlenych-teles/*

[14] Map Geocoded data with Gephi [online], gephi.org. [cit. duben 2014] Dostupné na: *http://gephi.org/tag/geolayout/*

A Uživatelská příručka

V této kapitole můžeme najít souhrn veškerých funkcí, které aplikace má. Vše je shrnuto v tabulce č. 1 a 2.

Tabulka 1: Seznam funkcí prováděných myší

Funkce	Akce
Označení/odznačení vrcholu/hrany	Levé tlačítko myši
Rotace grafu	Pohyb myši se stisknutým pravým tlačítkem myši
Posun grafu	Pohyb myši se stisknutým prostředním tlačítkem myši
Přiblížení grafu	Točení kolečkem myši

Tabulka 2: Seznam funkcí prováděných klávesnicí

Funkce	Klávesová zkratka
Klasické hrany	1
Šrafované hrany	3
Skrytí hran	4
Klasické hrany komunity	8
Šrafované hrany komunity	7
Zobrazení/skrytí popisků vrcholů	L
Zapnutí/vypnutí prolínání popisků vrcholů	K
Zapnutí/vypnutí módu komunit	M
Následující komunita	C
Předchozí komunita	X
Mód výběru objektu (hrana/vrchol)	J
Zobrazení/skrytí knihy	G
Zobrazení/skrytí vrcholů a hran nepatřících do komunity v módu komunit	V
Odznačení všech vrcholů	ESC
Posun označených vrcholů	Šipky $\uparrow$ $\downarrow$ $\rightarrow$ $\leftarrow$

B Adresářová struktura příloh

- Aplikace – projekt se zdrojovými kódy aplikace